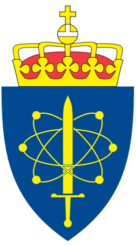


# FFI-RAPPORT

20/00002

# Optimization of training programs for competency-based training

## – COMFORT model description

Eirik Løhaugen Fjærbu

# Optimization of training programs for competency-based training

– COMFORT model description

Eirik Løhaugen Fjærbu

Norwegian Defence Research Establishment (FFI) 7 January 2020

**Keywords**
Modellering
Trening
Flygere
Kampfly
Java
Optimering

**FFI report**
20/00002

**Project number**
1449

**Elektronisk ISBN**
978-82-464-3250-2

**Approvers**
Stian Betten, *Research Manager*
Arne Petter Bartholsen, *Director of Research*

*The document is electronically approved and therefore has no handwritten signature.*

**Copyright**
© Norwegian Defence Research Establishment (FFI). The publication may be freely cited where the source is acknowledged.

# Summary

The Royal Norwegian Air Force (RNoAF) is in the process of phasing in new F-35 combat aircraft, which will take on a wide range of roles. Efficient pilot training is crucial in order to make full use of this investment. Simulator training will be an integrated part of the training program for the F-35 pilots. The simulators provide a means to better adapt the training to the needs of the pilots.

One possible approach to improve training efficiency is competency-based training, where training is planned according to competency requirements rather than requiring pilots to complete specific missions. Competency-based training is a promising approach that may improve the training outcome and reduce costs. At the same time, the scheduling and resource allocation can be challenging due to the high degree of flexibility required. In this report, we make use of optimization algorithms to assess these challenges.

We consider a regime where the set of missions that the pilots can carry out and the set of required competencies are given. Our aim is to minimize the total cost of the pilot training according to a given cost function, while ensuring that the pilots train each competency a given number of times. Two cases are considered: one where the training outcome of each mission is fixed and one where it can vary between different repetitions of the same mission.

Both problems are formulated as Constrained Optimization Problems (COPs). For the case where the training outcome varies, we compare two equivalent formulations. We implement a Java program named COmpetency-Mission Frequency Optimizer for Readiness Training (COMFORT) that solves the COPs using the software package OR-Tools. OR-Tools includes implementations of three different COP solution algorithms. We compare the three solution algorithms in terms of computation time for a set of example parameters and highlight some of their general strengths and weaknesses.

# Sammendrag

Luftforsvaret er i ferd med å fase inn nye kampfly av typen F-35, som skal fylle et vidt spekter av roller. Et effektivt treningsopplegg for pilotene er avgjørende for å få fullt utbytte av denne investeringen. Simulatortrening kommer til å bli en integrert del av treningsprogrammet til F-35-pilotene. Simulatorene gir muligheter for å tilpasse treningen bedre etter pilotenes behov.

En mulig framgangsmåte for å effektivisere treningsopplegget er kompetansebasert trening, hvor treningen planlegges ut fra kompetansekrav i stedet for krav om å gjennomføre spesifikke oppdrag. Kompetansebasert trening er en lovende framgangsmåte som kan forbedre treningsutbyttet og redusere kostnadene. Samtidig kan tidsplanleggingen og ressursallokeringen bli utfordrende på grunn av den store fleksibiliteten som kreves. I denne rapporten bruker vi optimeringsalgoritmer for å se på disse utfordringene.

Vi ser på et regime der det er kjent hvilke treningsoppdrag pilotene skal gjennomføre og hvilke ferdigheter de må opprettholde. Målet er å minimere kostnaden av treningen ut fra en gitt kostnadsfunksjon og samtidig sikre at pilotene trener på hver ferdighet et gitt antall ganger per år. Vi ser på to varianter av dette problemet: en hvor treningsutbyttet for hvert oppdrag er fast, og en hvor det kan variere mellom ulike repetisjoner av oppdraget.

Begge problemene formuleres som føringsbaserte optimeringsproblemer (COP-er). For tilfellet hvor treningsutbyttet kan variere, sammenligner vi to ekvivalente formuleringer. Vi implementerer et Java-verktøy kalt COmpetency-Mission Frequency Optimizer for Readiness Training (COMFORT) som løser COP-ene ved hjelp av programvarepakken OR-Tools. OR-Tools inneholder implementasjoner av tre ulike løsningsalgoritmer for COP-er. Vi sammenligner beregningstiden som brukes med de ulike algoritmene for et sett med eksempelparametre, og trekker fram noen generelle styrker og svakheter ved hver algoritme.

# Contents

**Summary** 3

**Sammendrag** 4

**1 Introduction** 7
- 1.1 Training Media 7
- 1.2 Continuation Training 7
- 1.3 Competency-Based Training 8
- 1.4 Outline 9

**2 Optimization and Simulation** 10
- 2.1 Constraint Satisfaction Problems 10
  - 2.1.1 Basic Algorithms 10
- 2.2 Linear Optimization 11
  - 2.2.1 Linear Programming 11
  - 2.2.2 Branch-and-Cut Algorithm 12
- 2.3 Discrete-Event Simulations 13
  - 2.3.1 Pilot Scheduling as a COP 14

**3 OR-Tools** 15
- 3.1 Constraint Solver 15
- 3.2 SAT Solver 15
- 3.3 IP Solvers 16

**4 COMFORT** 17
- 4.1 Baseline Problem 17
- 4.2 Cost Functions 18
- 4.3 Focus Competencies 19
  - 4.3.1 Individual Sorties 19
  - 4.3.2 Combinations of Competencies 20
- 4.4 Parameters and Return Values 21

**5 Implementation** 23
- 5.1 CP1 – Constraint Solver 23
- 5.2 SAT1 – SAT Solver 24
- 5.3 IP1 – IP Solver 25
- 5.4 CP2 – Individual Sorties 25
- 5.5 IP3 – Combinations of Competencies 26

**6 Example Parameters and Results** 27
- 6.1 Parameters 27

6.2 Baseline Problem 29
6.3 Focus-Competency Problem 30
6.4 Computation Time 31

**7 Discussion** 36
7.1 Solution Algorithms 36
7.2 Complexity 37
7.3 Improvements 37
7.4 Conclusion 38

**References** 39

# 1 Introduction

Training of pilots is crucial in order to utilize the full capabilities of combat aircraft, but requires a lot of time and resources. Therefore, efficient pilot training is of great importance. Combat readiness requires a high degree of competency over a wide range of skills, which must be maintained within the constrained pilot schedules. Furthermore, many of the necessary competencies can only be obtained through training in real combat aircraft (live training) [1], which is associated with significant costs and risks.

The Royal Norwegian Air Force (RNoAF) is in the process of acquiring new combat aircraft of the type F-35A CTOL Lightning II. These aircraft are capable of carrying out a wider range of tasks [2], and will fulfill a greater number of roles than the F-16A/B Fighting Falcons [3, 4], which the F-35s will replace. This will entail new competency requirements for the pilots, which in turn must be reflected in the pilot training [5].

The Norwegian Defence Research Establishment (FFI) has analyzed F-35 pilot training for a number of years, in support of RNoAF [5–13]. In this report, we present an optimization algorithm for use in pilot training. Before we describe the optimization algorithm, we will briefly discuss the use of simulators and competency-based approaches in pilot training.

## 1.1 Training Media

In the F-16 training program, simulator training only makes up a small fraction [1]. For the F-35 pilots, simulator training is an integrated part of the training program. In conjunction with the new combat aircraft, the RNoAF has acquired eight Full Mission Simulators (FMSs) that model the F-35 aircraft [14] for the purpose of pilot training. In contrast to the simulators used by F-16 pilots, the new FMSs will be connected in a network [1], allowing pilots to carry out tactical training missions. Additionally, opposing aircraft (red-air) can be modeled by Computer-Generated Forces (CGFs) or controlled via simpler simulators dedicated to red-air (RATS) [6]. Missions involving a large number of aircraft are difficult and costly to organize live, and are therefore particularly well suited for simulator training.

However, some missions are only suited for one training medium (i.e. live aircraft or simulator). For example, highly classified tactical maneuvers may be infeasible to execute live without risk of being detected by foreign surveillance [1]. Conversely, for some maneuvering exercises the strong g-forces experienced in the aircraft—which cannot be reproduced in the FMS simulators—are an essential part of the training [1].

## 1.2 Continuation Training

Combat aircraft pilots are assigned an operational classification as Combat Ready (CR), Limited-Combat Ready (LCR), or Non-Combat Ready (NCR) according to their competency, which determines whether a pilot can participate in combat operations. In order to retain their classification as combat ready, pilots must undergo continuation training. The squadron commanders are ultimately responsible for the assignment of the operational classification [15]. However, there are formal requirements for the types of missions the pilots should complete and which training medium should be used. These requirements specify the number of times each type of mission should be completed (sorties) within given time intervals (typically a year or 6 months) in order to be classified as CR. We will refer to such training regimes as frequency based.

The formal requirements for continuation training of F-16 pilots are given in the Combat Training Program (CTP) [16]. Based on the F-16 CTP and the new roles of the F-35 [4], a preliminary CTP was developed for the purpose of analyzing the training needs for F-35 pilots [17].

The simulation model TREFF was developed at FFI on the basis of this training program, in order to estimate the resources that are required for continuation training of F-35 pilots [5–7]. A number of studies have been performed using TREFF, where different scenarios and aspects of the training were analyzed [8–10]. Extending this work, a new simulation tool named TREFF2—which more accurately reflects how pilot training takes place—was developed in order to explore how training of F-35 pilots can be optimized [11–13].

## 1.3 Competency-Based Training

As we discussed above, combat aircraft pilots are required to maintain a large number of competencies. Competency-based training (CBT) is an approach that focuses on competencies as objectives rather than completion of missions. Here, we consider it as an alternative to the frequency-based approach discussed in section 1.2. CBT does not refer to a specific procedure, it can be implemented in several different ways. CBT is used in a range of different fields, both civilian and military [18, 19].

Several studies have shown that CBT can improve the learning outcome of pilot training compared to traditional approaches [20–22]. In particular, CBT has been successfully used for training in simulators [1, 20, 22]. Simulators are particularly well suited for CBT because they give complete control of flying conditions and the tactical environment, and because they allow for short sequences to be repeated quickly [1].

Determining the level of proficiency for each competency is an important part of a CBT regime. Ideally, performance data for each pilot should be obtained throughout the year, allowing instructors to adapt the training program to individual needs with a short response time. Such data can be obtained by automatic logging in the aircraft or simulator, self-evaluation, or evaluation by instructors. The data may be collected during regular training or in special missions designed to test proficiencies. In practice, the amount of data gathered will be limited due to the time and resources required to obtain the data, and by our ability to relate the data to specific competencies. If the available data is limited, the degree of proficiency may be estimated based on an expected competency retention interval [23]. In this case, the CBT program can be reduced into a training program with a set of missions and required frequencies like the CTP. However, this approach—where the training program is derived based on a set of competencies—may lead to significant changes to the contents of the training program.

The National Aerospace Laboratory (NLR) of the Netherlands has developed a method for designing such training programs, which is intended to be applied to several different types of aircraft including the F-35 [24]. Their CBT approach involves a hierarchy with a large number of low-level competencies and a smaller number of higher-level competencies [24, 25]. Training programs are designed by combining exercises that emulate complete missions (whole-task training) with smaller exercises that focus on specific low-level competencies (part-task training). The NLR has also identified a number of complexity factors, which are operating conditions that affect the execution of training missions. Examples include light- and weather conditions and technical issues with the aircraft. Complexity factors can alter the outcome of both training- and real operations significantly, and pilots may have to practice handling them.

## 1.4 Outline

In this report, we consider the relationship between the competency requirements and the repetition frequencies of the missions in a competency-based training regime. We start with a predetermined set of competency requirements and a set of training missions, where each mission can be used to practice a subset of the competencies. We consider a range of different methods to determine the optimal number of repetitions for each mission according to some cost function.

The remaining chapters are laid out as follows: In chapter 2, we describe some relevant models and mathematical problems. Section 2.1 and 2.2 introduce two classes of optimization problems, and outline some of the existing solution algorithms. Section 2.3 describes simulation-based tools developed at FFI in order to study F-35 pilot training, and the pilot scheduling used in these simulations. Chapter 3 introduces the open-source software package OR-Tools, which we use to solve optimization problems. In chapter 4, we formulate the optimization problem of missions and competencies mathematically, and define three different variants of the problem. We have created a number of different implementations of the optimization problem, corresponding to different variants and solution algorithms. These implementations are described in chapter 5, along with an overview of our source code. In chapter 6, we describe an example problem and its optimal solutions. We compare the results given by different solutions algorithms and the required computation time. These results and their implications are discussed in chapter 7.

# 2 Optimization and Simulation

Before we go any further, we will describe the mathematical formalism that we will use throughout the rest of the report. We will formulate our optimization problems as either Constrained Optimization Problems (COPs) or Integer Programs (IPs). These classes of optimization problems are discussed in section 2.1 and 2.2, respectively. We also include a short discussion on simulations used to study pilot training at FFI in section 2.3.

## 2.1 Constraint Satisfaction Problems

A Constraint Satisfaction Problem (CSP) is a mathematical problem defined in terms of three parts: a set of variables, the corresponding domains, and a set of constraints for the variables [26]. The goal is to assign a value to each variable from the corresponding domain so that all the constraints are satisfied. An assignment of a value to a variable is referred to as a label. A constraint is defined on a subset of the variables of the CSP, and specifies a set of allowed compound labels for the relevant variables. It is also assumed that all the variables have finite domains. Continuous domains such as position coordinates should therefore be reduced to finite domains, for instance by grouping the relevant positions into a finite number of regions.

In a CSP, all solutions are equivalent. However, it is often useful to define an objective function which assigns a number to each solution, and then determine the solution where the objective has the highest (or lowest) value. This type of problem is sometimes referred to as Constrained Optimization Problems (COPs). A COP can be solved iteratively by finding a solution to the corresponding CSP, adding a constraint requiring that the next solution is more optimal than the previous one, and repeating until there are no more solutions. All the problems that we solve in this work are formulated as COPs.

So far, we have not specified what the elements of the domains are, only that there is a finite number of elements. In practice, it is often useful to represent each element with an integer. We name the resulting integer variables $x_1, x_2, ..., x_n$, where $n$ is the number of variables. Each constraint $i$ can be specified in terms of a function $g_i$ that maps the integer variables onto a real number. If $g_i$ returns a negative number wherever constraint $i$ is satisfied, and a positive number otherwise, the constraint $i$ corresponds to

$$g_i(x_1, x_2, ..., x_n) \leq 0. \tag{2.1}$$

In many cases, the functions $g_i$ can be defined in terms of simple mathematical relations. The resulting problem is equivalent to the original CSP. For COPs, we also include an objective function $o(x_1, x_2, ..., x_n)$. In this form, COPs are often referred to as nonlinear integer programs [27].

### 2.1.1 Basic Algorithms

CSPs are a very general class of mathematical problems, but there exist general algorithms that can be used to determine one or all solutions given that enough time is available. We refer to implementations of such algorithms as general-purpose CSP solvers. There is a wide range of available solvers, but many of them are based on two basic algorithms: search with simple backtracking and problem reduction.

Conceptually, search with backtracking consists of choosing one variable at a time, trying to assign values to it and checking if any constraints are violated. If no constraints are violated, we choose a new variable and repeat the process. If a constraint is violated, we try a new value and

check the constraint again. If no value is allowed, we go back to the last variable that was assigned and try to assign it differently (backtracking). It is relatively easy to make an algorithm of this type which is guaranteed to find a solution eventually, but it is often challenging to define the order in which the computer chooses variables and values so that it finds a solution quickly.

Problem reduction refers to reducing the domains of variables and tightening constraints by removing values and compound labels that cannot lead to any solutions. For instance, if a value in a domain is forbidden by a constraint for all possible assignments of the other variables, it can be removed from the domain safely. Problem reduction reduces the size of the search space of the backtracking algorithm without removing any viable solutions. In many cases, it is useful to apply both problem reduction and a search algorithm to the same CSP in order to solve the problem more efficiently [26].

There are also several solution algorithms that apply only to specific types of CSPs, but which can be much more efficient than simple backtracking and problem reduction. We will consider one such case later on.

Another useful concept is soft constraints, which are conditions that we ideally want to satisfy, but where we might be interested in solutions where the constraint is broken. For example, this might be the case if we have a large number of constraints that cannot all be satisfied at once, but we want to satisfy as many as possible. In this nomenclature, the constraints from the CSP are called hard constraints. Soft constraints are included by adding a penalty to the objective function in a COP when the constraint is broken. The soft constraints are then prioritized through the optimization process according to the severity of the penalties.

## 2.2 Linear Optimization

If the constraint functions $g_i$ of equation 2.1 and the objective function $o$ are all linear, we refer to the problem as a (linear) Integer Program (IP). In this sense, IPs are a subclass of COPs. IPs are of great interest because there are much more efficient solution algorithms than the general ones outlined in section 2.1.1. IPs can be written on several equivalent forms, we define them as follows: Each variable $x_1, x_2, ..., x_n$ is an integer restricted to a finite interval $u_j \leq x_j \leq v_j$. Each constraint $i$ is specified by a linear combination of the variables and a boundary value $b_i$,

$$a_{i,1}x_1 + a_{i,2}x_2 + ... + a_{i,n}x_n \leq b_i. \quad (2.2)$$

The objective function $o$ is also given by a linear combination of the variables,

$$o(x_1, x_2, ..., x_n) = c_1x_1 + c_2x_2 + ... + c_nx_n. \quad (2.3)$$

The coefficients $a$, $b$, and $c$ can be any real numbers. We will outline an efficient solution algorithm for IPs in section 2.2.2. However, we first have to consider a closely related class of optimization problems known as Linear Programs (LPs).

### 2.2.1 Linear Programming

Linear Programs (LPs) take the same form as Integer Programs, but the variables are allowed to be any real numbers within a finite range, rather than being restricted to the integers. As in a COP, an objective function is maximized or minimized in an LP, while subject to a set of constraints. However, since the variables in LPs are real numbers, they do not have finite domains. Therefore,

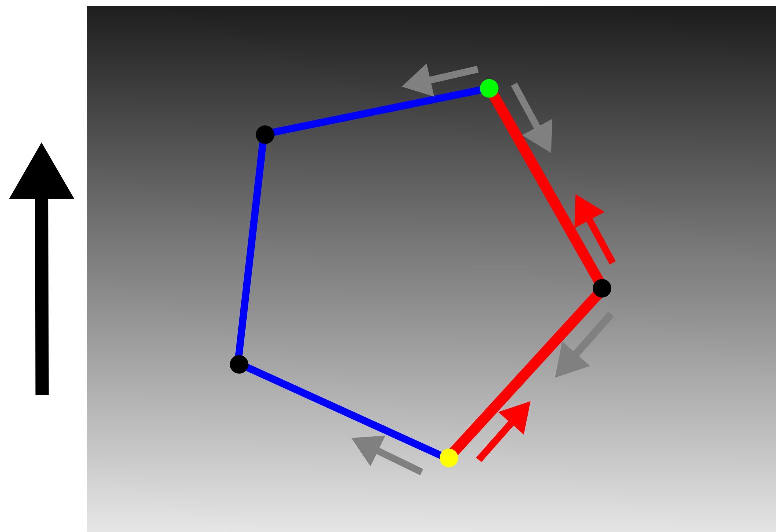


*Figure 2.1 Two-dimensional linear program consisting of five constraints (lines) and an objective (gradient and black arrow). The interior of the pentagon is the set of feasible solutions. The simplex algorithm traverses the vertices from the bottom (yellow) to the top (green) along the red line from one neighbor to another, always choosing the neighbor with the most optimal objective value (red arrows).*

linear programs are not COPs. Like integer programs, LPs may be defined in several different equivalent ways. We use the same formalism as for the integer programs defined above.

We illustrate an example of an LP with two variables in figure 2.1. For such two-dimensional LPs, each constraint corresponds to requiring that the solution lies on a given side of a straight line, and the objective function corresponds to the projection of the position onto an axis. The red and blue lines in figure 2.1 correspond to the constraints, and the objective function is the projection onto the vertical axis.

Since there are infinitely many values of each variable to consider, one might expect that an LP is more complicated to solve than a COP, but it turns out that the problem can be solved very efficiently. George Dantzig showed this by developing the simplex algorithm [28], which we will now outline. From looking at figure 2.1, one can convince oneself that in any two-dimensional LP, the optimal solution must occur at a corner of the feasible region. If there is more than one optimal solution, at least one of them must be at a corner. Thus, the relevant search space is already reduced to the finite set of vertices. It can be shown that this extends to higher-dimensional problems. Furthermore, it turns out that for any vertex of the feasible region except the optimal one, one of its neighboring vertices is always more optimal. The simplex algorithm traverses the vertices in this way until it reaches the optimal solution, see figure 2.1.

### 2.2.2 Branch-and-Cut Algorithm

As a strategy for solving IPs, one might naively try to solve the corresponding LP and look for an integer solution in the vicinity of the LP solution. However, this approach turns out to be insufficient, as the best integer solution can be very different from the best non-integer solution. Nevertheless, the LP solution does give us an upper (or lower) bound for the objective function, and it can be used to guide our search for an optimal solution.

The branch-and-bound- and cutting-plane algorithms are two well-known algorithms for solving IPs [29]. The simplex method is central to both approaches. The branch-and-cut algorithm combines the two approaches into one algorithm, which can be more efficient than either approach alone [30].

The branch-and-bound procedure divides the feasible region iteratively into smaller and smaller regions. For each region, one starts by finding the optimal solution of the corresponding LP with the simplex method. If the LP solution is integer-valued, we have found the IP solution for this region, so we move on to the next region. If the LP solution is less optimal than one of the IP solutions we have found before, we know that the region cannot contain the optimal IP solution, so we move on. If none of these are the case, we choose one of the non-integer variables and divide the region into two: one where that variable must be higher than for the LP solution and one where it must be smaller. The two regions are shrunk further by rounding the boundaries up and down to the nearest integers values, respectively. If one proceeds to split each regions into smaller ones in this way, the LP solution of some regions should eventually be integer-valued, so that less optimal regions can be disregarded.

In the cutting-plane algorithm, one starts by solving the LP problem using the simplex method. Based on the results of the simplex algorithm, it is possible to add a new constraint called a cutting plane, which is satisfied by all the feasible integer solutions, but not by the optimal LP solution. This process is repeated until the LP solution is integer-valued. The cutting-plane algorithm is usually slow compared to branch-and-bound [29]. However, cutting planes have been added as an additional step in the branch-and-bound procedure in order to give a more efficient method. This is referred to as a branch-and-cut algorithm [30].

## 2.3 Discrete-Event Simulations

The simulation models TREFF and TREFF2 were developed at FFI in order to analyze training of F-35 pilots [5–7, 11–13]. Both TREFF and TREFF2 are Discrete-Event Simulations (DESs), where all changes to the state of the system are described by a countable set of events [31]. The state of the system changes instantaneously at the time of an event, and remains constant between consecutive events. Therefore, it is sufficient to model the events in order to describe the whole range of time. Examples of events in TREFF are pilots arriving at work or starting a mission, and possible states for a pilot include flying a mission, being briefed, or being on holiday [7]. Both TREFF and TREFF2 are implemented using the simulation tool AnyLogic [32].

In order to carry out the simulations, we create schedules specifying which activity each pilot should be doing at any given time. Conditions such as weather, illness, and technical issues cannot be predicted far in advance, but can affect the pilot training schedule significantly. They are therefore treated as stochastic variables in the simulations. Therefore, the outcome of a simulation can vary between executions even for the same model. The pilot schedules often have to be modified due to such stochastic events, and are therefore created as part of the simulation.

In TREFF, at the start of each day, the scheduler prioritizes the pilots according to who has completed the fewest missions so far [6]. Then, it considers all pilots and all the missions that they have left to complete this year in the prioritized order. The scheduler determines whether the mission can be carried out live or in the simulator according to a flowchart with a set of conditions. When a mission is chosen, a new series of flowcharts is used to assign the remaining pilots to the exercise, and at what time of day the mission will take place.

### 2.3.1 Pilot Scheduling as a COP

Mathematically, the pilot training schedule can be represented as a set of variables with finite domains. The requirements that govern pilot training correspond to constraints on these variables. Thus, the scheduling problem can be formulated as a CSP. Here, additional requirements can be added to the schedule conveniently by adding new constraints, whereas the flow-chart approach in TREFF may require a lot of changes. In many cases it is useful to include an objective function, turning the scheduling problem into a COP. For instance, the objective function may represent the amount of time needed to complete a training program, or the amount of time a pilot is idle.

In TREFF2, the scheduling is performed by a separate module named SOFT [11]. In SOFT, the scheduling problem is formulated as a COP, with variables representing what each pilot is doing in a given time slot. SOFT also groups the pilots according to which formations they are qualified to lead, and accounts for these qualifications in the schedule. These qualifications are not accounted for in TREFF. Several properties of the schedule can be included in the cost function of the COP, such as whether the pilots complete too many or too few sorties of each type [12]. Rather than making a schedule for one day at a time, SOFT creates schedules for longer periods of time, which is also the case in actual squadrons. This also gives the scheduler more flexibility than the flowcharts used in TREFF. For instance, if a mission has to be flown on two different days, one might reduce the number of pilots taking part on the first day so that fewer pilots have to repeat the mission. In this way, the increased flexibility can be used to reduce the total number of sorties.

# 3 OR-Tools

OR-Tools is an open-source software suite for solving CSPs and optimization problems [33]. It contains a set of functions used to define a CSP, solve it, and read out the solutions. Four different versions of OR-Tools are available, for use with the programming languages C++, python, C#, and Java. We used the Java version for this work. Some parts of the functionality differs between the different programming languages. For instance, all the languages except Java allow for operator overloading, so that operators can interact with objects defined in OR-Tools.

The OR-Tools Java library defines a set of classes representing concepts such as variables, constraints, objectives, and solvers. The functions used to define and solve CSPs are methods associated with these classes. OR-Tools includes two general-purpose solvers for CSPs, which we refer to as the constraint solver and the SAT solver. Each of these solvers has its own set of classes and methods, with different names and functionality. Therefore, some work is required in order to switch between these two solvers. OR-Tools also provides a separate set of classes for defining and solving integer- and linear programs. Several different IP solvers can be called using the same set of classes and methods. One such solver—CBC—is included in OR-Tools, whereas others must be obtained separately.

## 3.1 Constraint Solver

The constraint solver has been part of OR-Tools for several years, and appears to have the most diverse set of functionality based on the examples we considered in this work. It allows the user to define and solve a wide range CSPs on the form given in section 2.1, using mathematical relations such as scalar products and inequalities to specify the constraints. The pilot-scheduling module SOFT is based on this solver [12]. In this work, we want to compare it to other CSP- and IP solvers, both in terms of functionality and performance.

The constraint solver uses a search algorithm with backtracking, see the discussion in section 2.1.1. The order in which variables are chosen and values from the domains are assigned are left as options for the user. These settings are specified using a *DecisionBuilder*-object [12]. This gives the user a lot of flexibility to optimize the solution algorithm and guide the solver towards the optimal solution. However, finding the optimal settings requires a lot of insight from the user and may be time-consuming.

For COPs, the constraint solver might compare all possible solutions of the corresponding CSP before it determines which is the optimal one. In order to find the optimal solution more efficiently, one can choose the order of the search so that the optimal solution will be found quickly. Then, one can stop the search before the solver has found all possible solutions. Again, this requires the user to know a lot about the problem beforehand. Finally, it may be beneficial to reduce the problem before passing it to the solver, for instance by removing values from the domains that cannot be part of any solutions, or adding constraints to remove redundant solutions.

## 3.2 SAT Solver

The SAT solver in OR-Tools is a new general-purpose CSP solver, which is recommended by the developers for most purposes [33]. Here, SAT refers to Boolean SATisfiability problems. This solver is intended to allow the user to specify a CSP in a similar way as the constraint solver, but at the same time solve the problem faster. In order to do so, the CSP given by the user is translated

into an SAT, in which all the variables can take only two values. The SAT is then solved using a Conflict-Driven Clause Learning (CDCL) algorithm [34, 35].

The CDCL algorithm is a form of search algorithm like the one described in section 2.1.1, but built specifically for problems involving Boolean variables. When any variable is assigned, the solver goes through all the conditions to see if any other variables follow from this assignment. Furthermore, the backtracking in the CDCL solver is implemented differently from the simple backtracking of section 2.1.1. If a conflict is found, rather than simply jumping back to the previous variable that was assigned and trying the next value, the solver determines which variables that caused the conflict and makes a new constraint for those. The solver then jumps back to the first variable that is part of the new constraint, which was not necessarily the last one that was assigned.

Since the search algorithm is not working with the same formulation of the CSP as the one given by the user, it is not straightforward for the user to specify the behavior of the search algorithm. The SAT solver does not require any user input regarding how the search is carried out. If the SAT solver is to outperform the constraint solver when the *DecisionBuilder* is well-configured, the CDCL algorithm has to be faster than simple backtracking inherently. We will compare the run time of the constraint solver and the SAT solver in section 6.4. The SAT solver may be easier to use than the constraint solver, especially for new users who have little experience with solving CSPs, because one does not have to configure the *DecisionBuilder*.

## 3.3 IP Solvers

The final set of classes in OR-Tools that we consider is built specifically for integer programs. It is possible to use these classes to call several different IP solvers, namely CBC [36] (Open-Source), GLPK [37] (Open-Source), SCIP [38] (Open-Source), and Gurobi [39] (Commercial). The CBC solver is included in OR-Tools, and is based on a branch-and-cut algorithm.

Since these classes are built specifically for IPs, the objective is specified using a function that sets one of the coefficients $c_{i,j}$ from equation 2.3 at a time. Similarly, the constraints are specified using functions that set the coefficients $a_{i,j}$ and the boundary $b_j$ of equation 2.2. Like the SAT-solver, the IP-solver does not require any user input to specify how the search should be carried out, it is sufficient to specify the problem at hand. For IPs formulated in the form of section 2.2, the IP solvers are expected to be much faster than general-purpose CSP solvers. Furthermore, since integer programs are NP-complete [40], any CSP can be represented by a set of IPs. However, the set of IPs may be harder to solve than the original CSP, even with efficient IP solvers.

# 4 COMFORT

As we discussed in section 1.3, competency-based training is an approach where training is planned according to a set of required competencies. COmpetency-Mission Frequency Optimizer for Readiness Training (COMFORT) is an analysis tool that determines optimal mission repetition rates subject to competency requirements. We consider a scenario where we want to make a training program consisting of a predetermined set of missions for pilot continuation training. We also assume that a set of competencies has been identified, and that there are requirements specifying how many times each competency should be practiced in a year. Our objective is finding an optimal number of repetitions for each mission according to some cost function, that also satisfies the competency requirements. We formulate the basis optimization problem more rigorously in section 4.1. Section 4.3 describes two alternative variants of the optimization problem that represent a slightly different situation than the basis problem.

The main objective of this work is to explore the available methods for such optimization, not to make specific recommendations for the training program. Therefore, we have created a set of example parameters. In particular, we have limited ourselves to 10 (high-level) competencies, whereas a real situation is likely to involve a higher number of competencies in a hierarchy. The example parameters are given in chapter 6.

The cost function determines which repetition rates are considered optimal. Several aspects of the training can be represented in the cost function, such as the monetary cost or the pilot overtime. The cost of a full training program can be estimated using a simulation tool such as TREFF2. However, simulating a full squadron for a year requires a significant amount of computation time, making it infeasible to simulate all possible combinations of missions. Instead, we estimate the cost function using an average cost per sortie for each mission. Estimating these averages will be difficult, and the estimated cost will not be completely accurate, but as we will see this simplification dramatically reduces the complexity of the optimization problem. We discuss the cost functions further in section 4.2.

## 4.1 Baseline Problem

The aim is to determine the number of sorties a pilot should complete for each mission and each training medium (i.e. live aircraft or simulator) in order to obtain the required competencies. We therefore define two vectors with $M$ variables each, $L_j$ and $S_j$, representing the number of times mission $j$ is repeated live and in the simulator, respectively. We make the domains of each variable finite by introducing an upper limit for the number of sorties for each mission in each medium, $Q$. The number of possible states depends strongly on the upper limit $Q$, so we want to choose as small a value as possible without excluding the optimal repetition rates.

A central assumption of COMFORT is that each mission has a predetermined set of competencies that pilots practice when they carry out the mission. For example, if a competency corresponds to performing a task, pilots only practice that competency in missions where that task is involved. These relations are represented as an $N$-by-$M$ matrix $A_{i,j}$, where $M$ is the number of missions and $N$ is the number of competencies. $A_{i,j}$ is equal to 1 if competency $i$ is practiced during mission $j$, and 0 otherwise. For now, the correspondence of competencies and missions is assumed to be the same for all sorties. Later, we will consider a case where the set of competencies practiced in a mission can vary between sorties.

The requirements for the number of repetitions of each competency is represented as a vector $B_i$

with $N$ numbers. The constraints representing the amount of practice needed for each competency take the form

$$B_i \leq A_{i,1}(L_1 + S_1) + A_{i,2}(L_2 + S_2) + \ldots + A_{i,M}(L_M + S_M). \tag{4.1}$$

If we identify $L_j = x_j$, $S_j = x_{j+M}$, $B_i = -b_i$, and $A_{i,j} = -a_{i,j} = -a_{i,j+M}$ for $i = 1, 2, \ldots, N$ and $j = 1, 2, \ldots, M$, these constraints are on the form of equation 2.2 for integer programs.

As we discussed above, we will estimate the cost of carrying out a training program by using an average cost for each sortie of a given mission. Live training requires a lot of time and resources, for instance fuel and maintenance of the aircraft. In this version of COMFORT, we assume that the costs of simulator training are much lower than for live training. We therefore set the cost of simulator training equal to zero. Including a cost for simulator training requires minimal modification of the model and source code as long as the cost is lower than for live training for all missions.

Denoting the average cost of a live sortie of mission $j$ per pilot by $C_j$, the total cost $O$ is

$$O = C_1 L_1 + C_2 L_2 + \ldots + C_M L_M. \tag{4.2}$$

The simplification we made by using an average cost for each sortie ensures that the cost function takes the form of equation 2.3, where $c_j = C_j$ and $c_{j+M} = 0$ for $j = 1, 2, \ldots, M$. Therefore, the COMFORT optimization problem is an integer program, see section 2.2. Since the branch-and-cut algorithm of section 2.2.2 is expected to be more efficient than the general-purpose methods of section 2.1.1, we should be able to solve this problem faster than similar problems with more complex cost functions.

Since simulator training does not entail any costs in this model, the solver will maximize the fraction of simulator training. However, as we saw in section 1.1, not all the necessary competencies can be obtained in simulators. Furthermore, some missions are better suited for training in simulators, and some are better suited for live training. In order to account for this, we include a restriction specifying that the ratio of simulator sorties to live sorties should be smaller than a given number for each mission. We specify the maximum simulator ratio in terms of two integers $F_j$ and $G_j$, giving

$$\frac{S_j}{L_j} \leq \frac{G_j}{F_j}. \tag{4.3}$$

If the cost of simulator training is higher than the cost of live training for the same mission, one should consider adding a minimum ratio of simulator training as well.

The number of possible solutions in the COMFORT model is relatively large, so there are a number of possible additional constraints that one might include. In this version, we added a minimum number of live sorties $K_j$ for each mission,

$$L_j \geq K_j, \tag{4.4}$$

in order to ensure that all missions are represented in the program.

## 4.2 Cost Functions

Finding a solution that satisfies the constraints of section 4.1 is relatively easy. For instance, one could find the competency $i$ where the required number of repetitions $B_i$ is highest, and make a program where $L_j = B_i$ for all $j$. However, such a program is likely to involve so many sorties that the pilots would be unable to complete it. COMFORT is useful because it allows us to find

repetition rates that are also optimal in some sense. But in order to get useful results, we need to define the cost function in a reasonable way.

Ideally, we would like to find a set of average costs $C_j$ for each live sortie of mission $j$ that gives a good approximation of the total cost for all possible training programs. However, in some cases it may be useful to tweak the average costs to give better results for repetition rates similar to the optimal ones, or add cost penalties for unwanted properties among the repetition rates (soft constraints). In many cases, the estimated cost will not give a good estimate of the actual cost of executing the training program, but still lead to repetition rates that are near optimal.

In order to facilitate combining multiple contributions to the cost function, we split the total cost into $Z$ components denoted $P1$, $P2$, ... ,$PZ$. Each component $Pn$ is defined by a set of average costs for each mission $C_j^{Pn}$. All the costs are assumed to be greater than or equal to zero, and we normalize each component cost function to be equal to one for the most costly mission. We combine these components into the total cost $C_j$ using weighting coefficients $w_{P1}$, $w_{P2}$, ..., $w_{PZ}$,

$$C_j = \frac{1}{w_{P1} + w_{P2} + \ldots + w_{PZ}} \left( w_{P1} C_j^{P1} + w_{P2} C_j^{P2} + \ldots + w_{PZ} C_j^{PZ} \right). \quad (4.5)$$

As long as the weighting coefficients are all positive, the total cost $C_j$ will be between 0 and 1 for all missions $j$.

## 4.3 Focus Competencies

For some complex missions, a large number of competencies are involved, which the pilots could practice during that mission if they dedicate some time to it. However, practicing all the competencies in a single sortie may take too much time or make the situation too complex for inexperienced pilots to handle. We consider a situation where the pilots select a few competencies to focus on during each sortie, and prioritize these competencies when they plan and carry out the mission. Since multiple different choices of focus competencies are possible, all the sorties with the same mission are not equivalent anymore. Therefore, the baseline COMFORT problem of section 4.1 must be reformulated to reflect the new situation.

We introduce an upper limit $D_j$ to the number of competencies practiced in each sortie of mission $j$. In principle, the number of focus competencies in a sortie could be lower than $D_j$, but in our model this cannot lead to any cost reductions. Therefore, we will assume that the number of competencies practiced in each sortie is equal to $D_j$. We assume that the cost function and the constraints take the same form as in section 4.1, except the constraint of equation 4.1. This constraint enforces the competency requirements, and must be reformulated to account for the focus competencies. We also assume that the set of competencies that are possible to practice in each mission is given by the matrix $A_{i,j}$, which takes the same form as in section 4.1.

There are multiple ways in which this version of the COMFORT problem can be formulated as a COP. In order to better illustrate the use of the different solvers in OR-Tools, we will describe two such variants. This will also allow us to show that reformulating an optimization problem can greatly reduce the required computation time without loss of generality.

### 4.3.1 Individual Sorties

Since the sorties for the same mission are not equivalent, the variables $L_j$ and $S_j$ are not sufficient to describe what the pilots practice for the focus-competency version of the problem. One possible

way to represent the new rates is to make a list of $K$ sorties and a set of variables describing each sortie. A benefit of this representation is that the solver does not have to consider solutions involving more than $K$ sorties, whereas for the baseline formulation of the problem there are solutions with up to $2 \cdot M \cdot Q$ sorties. For each competency $i$ and each sortie $k$, we introduce a Boolean variable $U_{i,k}$ which is equal to 1 if competency $i$ is a focus competency in sortie $k$ and 0 otherwise.

For each sortie $k$, we introduce a variable $W_k$ representing which mission is carried out in which medium, which takes a value between 0 and $2M$. Here, $W_k = j$ if mission $j$ is carried out live during sortie $k$, and $W_k = j + M$ if mission $j$ is carried out in a simulator. In order to allow the solver to determine the optimal value of the total number of sorties, we define $W_k = 0$ to represent that sortie $k$ is not used for training. The variables $L_j$ and $S_j$, which we defined in section 4.1, are determined from $W_k$ by counting the occurrences of $W_k = j$ and $W_k = j + M$.

In order to specify which competencies can be practiced during each mission, we define an $(N)$-by-$(2M + 1)$-matrix $A'_{i,j}$ such that $A'_{i,j} = A_{i,j}$ and $A'_{i,j+M} = A_{i,j}$ for $j = 1, 2, ..., M$. For each competency $i$ and sortie $k$, we add a constraint

$$A'_{i,W_k} \geq U_{i,k}, \tag{4.6}$$

ensuring that $U_{i,k}$ can only be equal to 1 if competency $i$ can be practiced during sortie $k$. Since our competencies must be obtained by carrying out missions, we set $A'_{i,0} = 0$ for all $i$. Similarly, we define a $(2M + 1)$-vector $D'_j$ so that $D'_j = D_j$, $D'_{j+M} = D_j$, and $D'_0 = 0$ for $j = 1, 2, ..., M$. We add a constraint

$$D'_{W_k} = U_{1,k} + U_{2,k} + ... + U_{N,k} \tag{4.7}$$

for each sortie $k$, specifying that the number of competencies practiced in sortie $k$ is equal to the upper limit for the corresponding mission.

The competency requirements of equation 4.1 now take the simple form

$$B_i \leq U_{i,1} + U_{i,2} + ... + U_{i,K}. \tag{4.8}$$

Since we have determined the values of $L_j$ and $S_j$, the remaining constraints take the same form as in section 4.1. Similarly, the cost function $O$ is on the form of equation 4.2.

The order in which the missions are carried out is not important in our model, so the individual-sortie approach will give a lot of different states which are all equivalent. This symmetry greatly increases the size of the search space that the solver has to go through in order to determine which state is optimal. In order to avoid going through many equivalent parts of the search space in a symmetric model, one can add symmetry-breaking constraints that reduce the redundancy [41]. One should ensure that for all states that are excluded by the symmetry-breaking constraints, at least one equivalent state is allowed. For example, we might insist that mission $j - 1$ cannot appear in the list of sorties after mission $j$ for all $j = 2, 3, ..., M$, ensuring that the missions appear in order. For every possible list of sorties, there is an equivalent ordered list that satisfies the symmetry-breaking constraints, which can be found by reordering the sorties. Symmetry-breaking constraints will not necessarily reduce the search time to find the first solution to satisfy all constraints, since the number of such feasible solutions is reduced. However, the symmetry-breaking constraints should reduce the search time required to go through all possible solutions.

### 4.3.2 Combinations of Competencies

For each mission $j$, there is a finite number ($Y_j$) of possible combinations of competencies in a single sortie. As an alternative approach to the focus-competency optimization problem, we consider each

of these combinations as a separate mission. This gives an optimization problem which is similar to the baseline problem, but where the effective number of missions is $\tilde{M} = Y_1 + Y_2 + \ldots + Y_M$. An advantage of this approach is that the set of possible combinations of competencies is built into the problem before we call the solver. By contrast, for the individual-sortie optimization problem, the solver has to work out the allowed combinations again each time it assigns a new mission to a sortie.

In the combinations-of-competencies variant of the optimization problem, the variables specify only the number of repetitions of each combination, and not the order in which they are executed. Therefore, each set of equivalent states from the individual-sortie variant is represented by a single state, so the search space is reduced drastically. Reformulating the model to avoid the symmetry in this way can be beneficial compared to adding symmetry-breaking constraints, since enforcing the constraints requires computation time in itself. However, the search space now includes states with up to $2 \cdot \tilde{M} \cdot \tilde{Q}$ sorties, where $\tilde{Q}$ is the maximum number of sorties per medium for each combination, which is typically larger than the length of the list of sorties.

Before we solve the main problem, we determine the set of possible combinations of competencies in a sortie for each mission. This problem can be defined as a simple CSP, and solved using OR-Tools. The number of possible combinations for mission $j$ is given by

$$Y_j = \binom{H_j}{D_j} = \frac{H_j!}{D_j!\,(H_j - D_j)!}, \tag{4.9}$$

where $H_j = A_{1,j} + A_{2,j} + \ldots + A_{N,j}$ is the number of competencies that are possible to practice in mission $j$. The combinations of competencies are represented by $A_{i,j,k}$, where $k = 1, 2, \ldots, Y_j$ indexes the possible combinations for mission $j$. Here, $A_{i,j,k}$ equals 1 if competency $i$ is part of combination $k$ of mission $j$, and 0 otherwise.

The main variables in this variant of the optimization problem are $L_{j,k}$ and $S_{j,k}$, representing the number of live- and simulator sorties of combination $k$ of mission $j$, respectively. The requirements for the amount of practice for each competency now take the form

$$\begin{aligned} B_i \le & \sum_{j,k} A_{i,j,k}(L_{j,k} + S_{j,k}) \\ = \; & A_{i,1,1}(L_{1,1} + S_{1,1}) + \ldots + A_{i,1,Y_1}(L_{1,Y_1} + S_{1,Y_1}) \\ & + \ldots \\ & + A_{i,N,1}(L_{N,1} + S_{N,1}) + \ldots + A_{i,N,Y_N}(L_{N,Y_N} + S_{N,Y_N}), \end{aligned} \tag{4.10}$$

where $\sum$ represents a sum.

The variables $L_j$ and $S_j$ are defined as before, and are related to the new variables via the constraints

$$L_j = L_{j,1} + L_{j,2} + \ldots + L_{j,Y_j}, \text{ and } S_j = S_{j,1} + S_{j,2} + \ldots + S_{j,Y_j}. \tag{4.11}$$

We use the expressions of equation 4.4 and 4.3 for the remaining constraints, and equation 4.2 for the cost function.

## 4.4 Parameters and Return Values

Before we move onto the implementation, we summarize what information is needed to define a COMFORT optimization problem and its optimal solution. We start with the baseline problem, and then discuss the differences with the focus-competency problem. The required parameters of the baseline problem are

- Whether each competency is practiced in each mission ($A_{i,j}$)
- Value of each component cost for each mission ($C_j^{Pn}$)
- Weights for each component cost ($w_{Pn}$)
- Required number of repetitions of each competency ($B_i$)
- Maximum allowed ratio of simulator training for each mission ($F_j$ and $G_j$)
- Minimum number of live repetitions of each mission ($K_j$)

In practice, the maximum number of repetitions $Q$ must be specified in order to run the program, but as long as $Q$ is large enough it will not affect the optimal solution.

The solution of the optimization problem is specified in terms of the number repetitions of each mission live ($L_j$) and in the simulator ($S_j$). We emphasize that we do not determine a full pilot schedule, only the number of repetitions of missions that a pilot should carry out. This greatly reduces the complexity of the optimization problem and the amount of input parameters required.

All the parameters that were required for the baseline problem are also required for the focus-competency problem. Additionally, we have to specify the maximum number of competencies practiced in a single sortie for each mission ($D_j$). As with $Q$, the parameters $K$ from the individual-sortie problem and $\tilde{Q}$ from the combination-of-competency problem will not affect the optimal solution as long as they are large enough. We describe the solution of the focus-competency optimization problem in terms of the number of repetitions of each combination of competencies for each mission live ($L_{j,k}$) and in the simulator ($S_{j,k}$). These results are sufficient to derive the number of repetitions for each mission ($L_j$ and $S_j$), but not vice versa.

# 5 Implementation

COMFORT is mainly implemented in Java. However, a number of Python scripts that generate figures using the package matplotlib are included. For each combination of a variant of the optimization problem and a solver from OR-Tools, we implemented a class containing a static method named *optimize*. The *optimize*-method specifies the optimization problem to the solver, calls the solver, and reads out the results. We refer to these classes as variant implementations of the COMFORT model. Additionally, COMFORT includes a number of common classes containing code that is used by all the variant implementations. In this section, we give an overview of the functionality of the common classes.

In the following sections, we give an outline of the COMFORT variant implementations that are relevant to our discussion. Two additional variant implementations called SAT2 and IP2 are provided in COMFORT, but not documented here. SAT2 uses the SAT solver from OR-Tools, whereas IP2 uses the IP solver CBC. Both SAT2 and IP2 implement an individual-sortie variant of the optimization problem, see section 4.3.1.

COMFORT contains three executable classes: *Solve*, *Basis*, and *Barcharts*, which contain static *main*-methods. The *Solve*-class solves a problem and prints the results to a terminal, the *Basis*-class generates a set of figures based on the input parameters, and the *Barcharts*-class solves a problem and generates figures representing the results. Some of the parameters that are most likely to be modified by the user are specified in the *Solve*- and *Barcharts*- classes. This includes the maximum number of repetitions $Q$ or $\tilde{Q}$, the cost-function weights $w_S$, $w_R$, and $w_O$, and a number $V$ used to convert the floating-point cost into integers. We obtain an integer cost function by multiplying all costs by a large number $V$ and rounding down to the nearest integer.

The remaining input parameters are provided by classes that implement the *Data*-interface. The *Data*-interface specifies that these classes must contain a set of methods that provide the input parameters listed in section 4.4. A class named *ExampleData* that implements the *Data*-interface is included in COMFORT. The parameter values provided by the *ExampleData*-class are presented in section 6.1. The *Results*-class stores the output data from the solvers in the form of arrays of integers and floating-point numbers. This class contains three different constructors, which are used to translate solutions from the three variants of the COMFORT optimization problem into a common form. COMFORT also includes a *Print*-class, which contains a static function that prints the repetition rates to the terminal in a human-readable form. The *Plotting*-class contains four static functions used to write data to *.txt*-files and call Python scripts.

The control flow for the *Solve*-class is illustrated in figure 5.1. The diagram for the *Barcharts*-class is similar, but includes additional calls to the *Plotting*-class. The control flow for the *Basis*-class is simple: it calls the constructor of the *ExampleData*-class and some methods from the *Plotting*-class. Note that the *Solve*- and *Barcharts*- classes can use any implementation of the *Data*-interface, whereas the *Basis*-class requires an instance of the *ExampleData*-class.

## 5.1 CP1 – Constraint Solver

The CP1 variant implementation solves the baseline problem using the constraint solver, see sections 3.1 and 4.1. We call the constraint solver using the *constraintsolver*-package from OR-Tools. The *Solver*-class is instantiated first. Most of the calls we make to OR-Tools are methods of the *Solver*-object. We represent the variables $L_j$, $S_j$, and $T_j$ using *IntVar*-objects.

The *optimize*-function takes a *Data*-object as an argument. Since the constraint solver requires

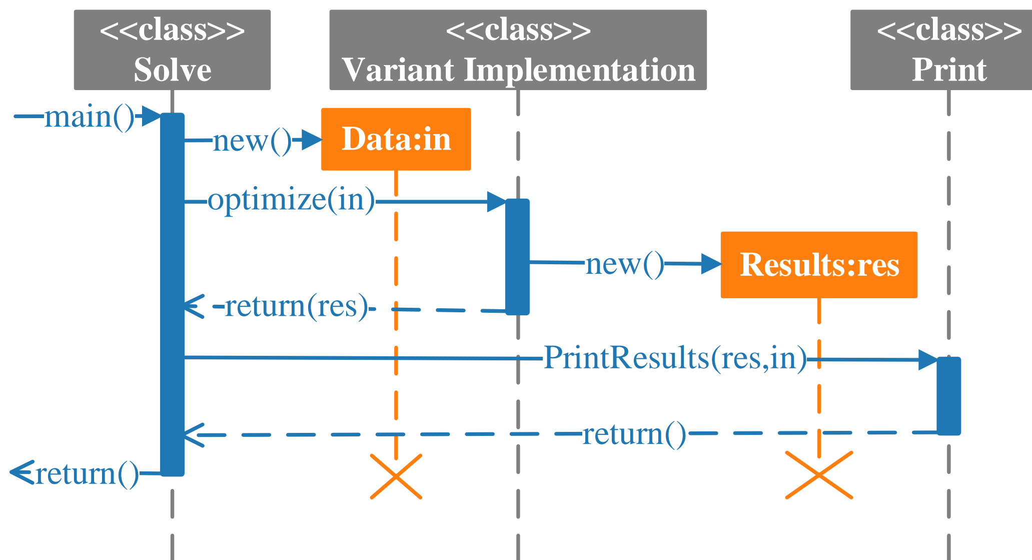



*Figure 5.1 Sequence diagram for the main-function of the Solve-class. The grey and orange rectangles are classes and objects, respectively. The blue arrows and bars illustrate static methods and constructors. The class Variant Implementation can be either CP1, CP2, IP1, IP3, or SAT1.*

integer parameters, the cost scale $V$ should be a large integer. The rescaled costs $C_j$ are read out from the *Data*-object in integer form. Next, we generate an *OptimizeVar*-object, specifying that we want to minimize the product of $C_j$ and $L_j$, see equation 4.2.

The *Solver*-object has a wide range of methods for implementing constraints. Each of these methods returns a *Constraint*-object, which has to be passed back to the solver using the *addConstraint*-method. We add constraints corresponding to equation 4.1, 4.3, and 4.4. Additionally, we add constraints specifying that the total variable $T_j$ is equal to the sum of $L_j$ and $S_j$.

Next, we use the *Solver*-object to create a *SolutionCollector*-object and a *DecisionBuilder*-object. The *SolutionCollector* is used to store the results found by the solver, and the *DecisionBuilder* assigns values to the variables $L_j$ and $S_j$ during the search. We specify that $L_1$ and $S_1$ should be assigned first, then $L_2$ and $S_2$, and so on. The *DecisionBuilder* is also instructed to try the possible values for each variable in ascending order. We call the constraint solver using the *newSearch*-method, and find new solutions iteratively using the *nextSolution*-method. We keep calling the *nextSolution*-method until it finds the optimal solution or a user-defined time limit has elapsed. Finally, the values for $L_j$ and $S_j$ representing the best solution found are read out and used to create a *Results*-object.

## 5.2 SAT1 – SAT Solver

The SAT1 variant implementation solves the same variant of the COMFORT problem as CP1, in an almost completely analogous way. However, we use the SAT solver provided in the *SAT*-package from OR-Tools instead of the constraint solver. In most cases, the classes and methods provided by the *SAT*-package behave analogously to those in the *constraintsolver*-package. One minor difference is that two classes, *CPModel* and *CpSolver*, share the role of the *Solver*-class from the

*constraintsolver*-package. Also, *Constraint*-objects are applied by default without the need to call a method like *addConstraint*. As for the constraint solver, the classes representing variables and constraints in the *SAT*-package are named *IntVar* and *Constraint*. This may cause name conflicts if both packages are used in the same method.

As we discussed in section 3.2, the SAT solver does not require us to specify the behavior of the search algorithm, so there is no equivalent of the *DecisionBuilder*-class. We solve the optimization problem with the *solve*-method of the *CpSolver*-object, which searches until the optimal solution is found. When an optimal solution is determined, it is retrieved via the *CpSolver*-object. We use the resulting values of $L_j$ and $S_j$ to create a *Results*-object as for the constraint solver.

## 5.3 IP1 – IP Solver

As we saw in section 4.1, the baseline COMFORT optimization problem can be written as an Integer Program (IP). The IP1 variant implementation solves it using the *linearsolver*-package of OR-Tools, analogously to CP1 and SAT1. In integer programs, each constraint corresponds to a linear combination of variables, and the same goes for the cost function. In the *linearsolver*-package, the variables, constraints, and cost functions are represented by *MPVariable*-, *MPConstraint*-, and *MPObjective*-objects, respectively. The *MPSolver*-class takes the role of the *Solver*-class of the *constraintsolver*-package. We specify that we want to use the CBC solver when we instantiate the *MPSolver*-class, see section 3.3.

When creating an *MPConstraint*-object, we specify an upper and a lower boundary value. Then, we specify coefficients for each *MPVariable* in the linear combination, one at a time. The cost coefficients for the IP solver are specified as floating-point numbers, and we set the cost scale $V$ equal to 1. Like the SAT solver, the CBC solver requires no user input regarding how the search should be carried out. We call the solver using the *solve*-method of the *MPSolver*-object, which searches for solutions until the optimal solution is found or an optional time limit has elapsed. The optimal solution is read out using methods associated with the *MPVariable*-objects themselves, and used to create a *Results*-object in the same way as before.

## 5.4 CP2 – Individual Sorties

The CP2 variant implementation solves the individual-sortie variant of the optimization problem using the *constraintsolver*-package from OR-Tools. The set of classes and methods used is the same as for the CP1 variant implementation of section 5.1, but the set of variables and constraints that we implement is different.

We start by defining *IntVar*-objects to represent the variables $W_k$, $U_{i,k}$, $L_j$, $S_j$, and $T_j$ described in section 4.3.1. Using the *makeDistribute*-method, we add constraints ensuring that $L_j$ and $S_j$ are equal to the number of occurrences of $j$ and $j + M$ among the variables $W_k$. Next, we create *IntVar*-objects representing $A'_{i,W_k}$ and $D'_{W_k}$ using the *makeElement*-method. These are used to implement the constraints of equation 4.6 and 4.7. The remaining constraints and the cost function are implemented in the same way as in section 5.1.

We create a *DecisionBuilder*-object, and specify that it should assign values to the variables $W_k$ and $U_{i,k}$ for one sortie ($k$) at a time. For each sortie $k$, the *DecisionBuilder* assigns a value to $W_k$ first, then $U_{1,k}$, $U_{2,k}$, and so on. The solver tries possible values for each variable in ascending order. A *SolutionCollector*-object is created, and instructed to store the values of $W_k$ and $U_{i,k}$ for the best solutions found. We call the constraint solver and search for solutions one at a time as in

section 5.1. Finally, we read out the values of $W_k$ and $U_{i,k}$ from the *SolutionCollector*, and pass them to the *Results*-object.

## 5.5 IP3 – Combinations of Competencies

Finally, we include an implementation of the combinations-of-competencies variant of the optimization problem. The IP3 variant implementation uses the constraint solver to determine the relevant combinations and the *linearsolver*-package to solve the main optimization problem. We use the constraint solver to find the combinations of competencies because the *SolutionCollector*-class provides a convenient way to read out all solutions of a CSP. For each mission, we define a CSP representing the possible combinations of competencies. We create separate *Solver*-, *IntVar*-, *DecisionBuilder*-, and *SolutionCollector*-objects for each CSP. Then, we solve each CSP using the *solve*-method, and read out all the solutions to an integer array.

We introduce a new index $l$ which indexes the combinations of competencies for all missions. The total number of combinations is $\tilde{M} = Y_1 + Y_2 + \ldots + Y_N$. For combination $k$ of mission $j$,

$$l = Y_1 + Y_2 + \ldots + Y_{j-1} + k. \tag{5.1}$$

The combinations of competencies are represented by an $N$-by-$\tilde{M}$ integer array $\tilde{A}_{i,l} = A_{i,j,k}$. See section 4.3.2 for the definition of $A_{i,j,k}$.

Next, we create *MPVariable*-objects representing $L_l = L_{j,k}$ and $S_l = S_{j,k}$, and set boundaries of 0 and $\tilde{Q}$ for each variable. We create an *MPObjective*-object which is to be minimized, and set the coefficient of each live-training variable $L_l$ equal to the corresponding cost $C_j$. For the competency requirements, we create an *MPConstraint*-object for each competency $i$ with a lower boundary of $B_i$. For each competency $i$ and combination $l$, we set the coefficients of both variables $L_l$ and $S_l$ in constraint $i$ equal to $\tilde{A}_{i,l}$.

The constraints of equation 4.3 and 4.4 refer to the number of sorties per mission rather than per combination of competencies. Here, we create one *MPConstraint*-object for each mission $j$, and set equal coefficients for all combinations $l$ from mission $j$. We call the CBC solver using the *solve*-method of the *MPSolver*-object as in section 5.3. We then read out the values of $L_{j,k}$ and $S_{j,k}$ for the optimal solution and pass them to the *Results*-object.

# 6 Example Parameters and Results

In order to compare the different solvers from OR-Tools and the different variants of the optimization problem, we have created a set of example parameters for which we calculate optimal repetition rates. In COMFORT, the example data is contained in the *ExampleData*-class. The *ExampleData*-class implements the *Data*-interface and contains all the input values listed in section 4.4 for the focus-competency problem.

As we noted in chapter 4, our example case is not intended to represent a real situation, and we therefore label the missions with characters $A$ to $Y$ and the competencies as C1 to C9. The example parameters are given in section 6.1. We present solutions of the basis problem and the focus-competency problem for the example parameters in sections 6.2 and 6.3. Finally, in section 6.4, we specify the computation time used by each variant implementation for the example parameters.

## 6.1 Parameters

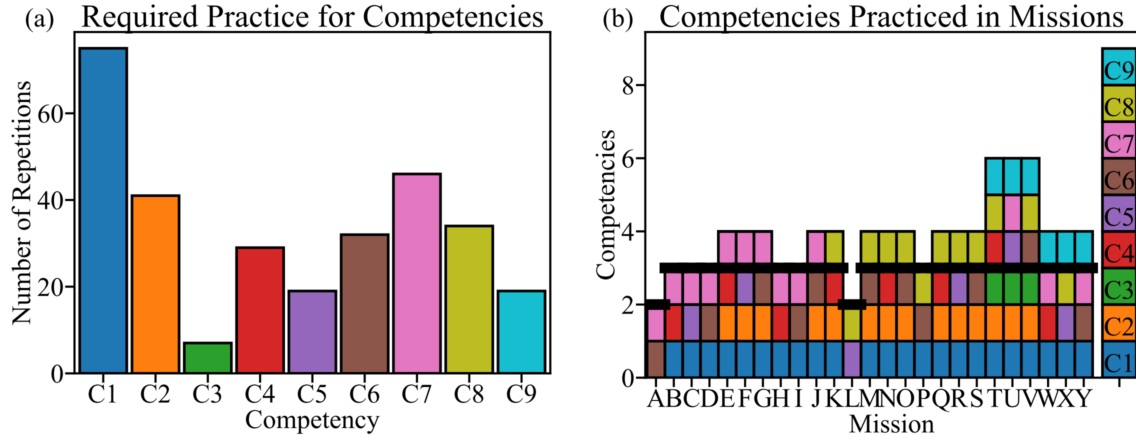



*Figure 6.1 Required number of times $B_i$ that each pilot must practice each competency (a), and the set of competencies that are possible to practice in each mission (b). The competencies are identified by colors. The maximum number of competencies that can be practiced in each sortie in the focus-competency version of the problem is indicated with black lines in (b).*

The required amount of practice for each competency $B_i$ is the central constraint that ensures that the pilots get sufficient training. We illustrate this constraint in figure 6.1 (a). Figure 6.1 (b) shows which missions can be used to practice to each competency, which is given by the matrix $A_{i,j}$. Missions T, U, and V are the most complex missions, and involve more competencies than the others. Competency C1 is a basic competency that the pilots can practice in most of the missions, whereas competency C3 is associated with the complex missions T, U, and V. The minimum number of live sorties $K_j$ is equal to 1 for all missions $j$.

For the cost function, we include three components $C_j^S$, $C_j^R$, and $C_j^O$ representing common costs associated with every sortie, additional costs associated with red-air, and additional organizational costs for complex missions. We normalize each of these component cost functions so that the cost of the most expensive mission is always equal to one. The sortie cost $C_j^S$ corresponds to the costs that are common to all missions, and is therefore simply equal to one for every mission.

In the current training regime [16, 42], taking part in a live training mission as opposing air forces (red-air) does not count towards all of the frequency requirements. Analogously, we expect that the learning outcome for the red-air pilots in F-35 will be limited, and therefore add an additional cost to missions where the fraction of pilots taking part as red-air is high. For simplicity, we assume that the number of pilots taking part in each mission is the same for all sorties, see figure 6.2. For this example data, we simply define the red-air cost to be proportional to the number of pilots on the red side divided by the number of pilots on the blue side. The normalized red-air cost coefficients are shown in figure 6.3 (a).

Finally, missions involving a large number of pilots and aircraft are typically harder to incorporate in a schedule than smaller ones. Also, the chance of unplanned absence among the pilots or technical issues with a plane increases the more pilots and planes are involved. The organizational component cost function is meant to reflect such reductions in training efficiency associated with complex missions. In the example data, the organizational cost is proportional to the total number of pilots taking part in each sortie, see figure 6.3 (b). As expected, the complex missions T, U, and V are the most expensive ones to organize.

We combine the components costs into a total cost $C_j$ using three weighting coefficients $w_S = 1.0$, $w_R = 0.5$, and $w_O = 0.5$,

$$C_j = \frac{1}{w_S + w_R + w_O}\left(w_S C_j^S + w_R C_j^R + w_O C_j^O\right). \tag{6.1}$$

As we discussed in section 4.2, a good choice of cost function is crucial in order to obtain useful results. In real situations, the component costs and the weighting coefficients should be chosen with care. Since sorties in the simulator do not incur any costs in our model, missions where a high percentage of the sorties can be carried out in the simulator are cheaper to organize overall. The maximum allowed ratio of simulator to live training $G_j/F_j$ is shown in figure 6.2. Missions where

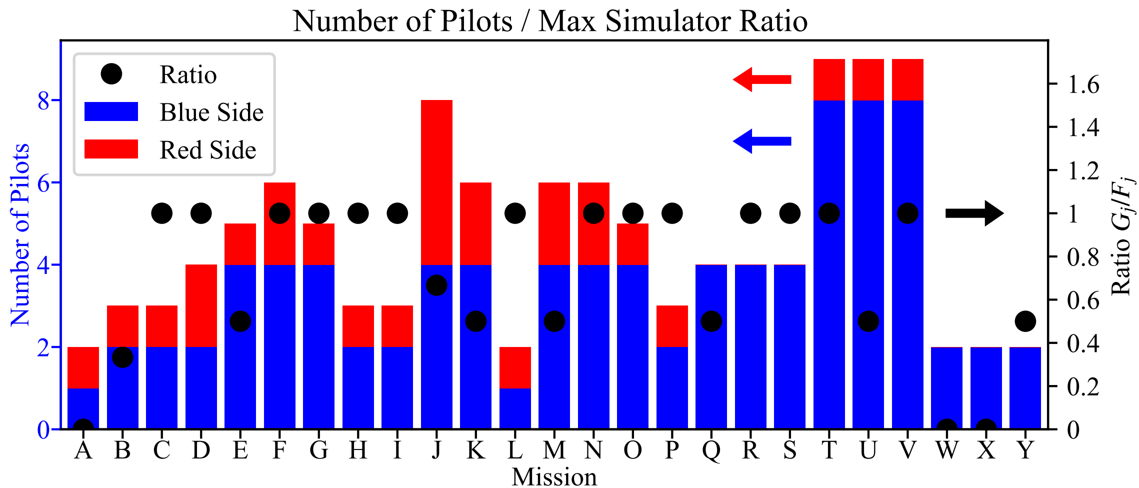



*Figure 6.2 Number of pilots on the blue side (blue) and red side (red) in each mission. These values are not part of the COMFORT optimization problem, but are used to estimate the red-air and organizational component cost functions. The maximum allowed ratio of simulator sorties to live sorties $G_j/F_j$ is superimposed as black dots. For the example parameters, the ratio $G_j/F_j$ is never larger than 1, but this is not a requirement of the model.*

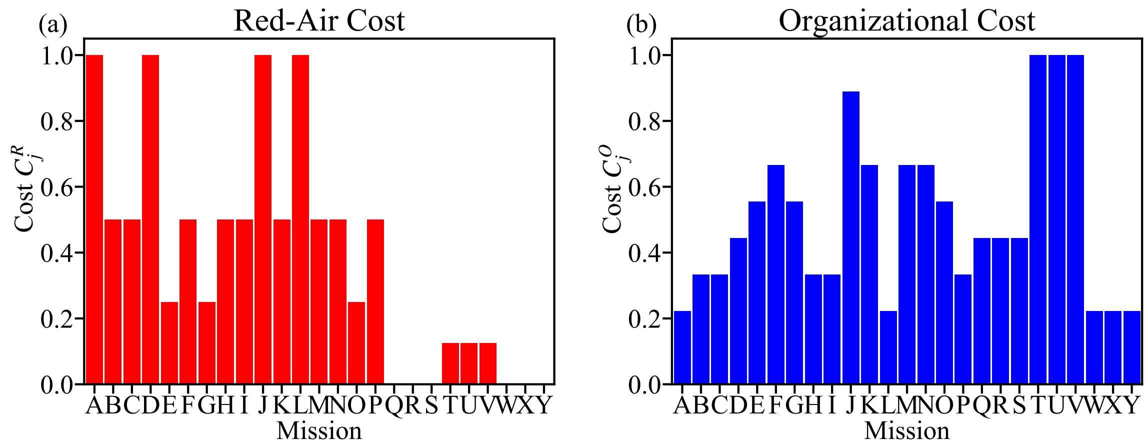



*Figure 6.3* *Red-air (a) and organizational (b) component cost functions $C_j^R$ and $C_j^O$, normalized to the range* 0 *to* 1*. The red-air cost is indicated by the red bars, and the blue bars show the organizational cost.*

the red-air and organizational costs are low and the simulator ratio $G_j/F_j$ is high—such as missions H, R, and S—give the lowest total cost per sortie.

## 6.2 Baseline Problem

For the baseline problem, both the SAT1- and IP1 variant implementations return optimal solutions within a short time. The solutions returned by SAT1 and IP1 may differ slightly because there may be multiple optimal solutions with the same cost, and because the cost coefficients are converted to integers in SAT1. In this section, we illustrate the optimal repetition rates returned by the IP solver in the form of bar charts.

The repetition rates are represented by the numbers of live- and simulator sorties $L_j$ and $S_j$ for each mission, see figure 6.4. In total, this solution consists of 47 live sorties and 33 simulator sorties. A large fraction of the training is made up of a few missions (H, R, S, and Y). These missions have relatively low costs and high allowed ratios of simulator training, see figures 6.3 and 6.2.

We also want to illustrate that the repetition rates of figure 6.4 actually satisfy the requirements for the amount of practice for each competency. In figure 6.5, we illustrate these requirements and show how the learning outcomes from the missions add up to fulfill them. As one would expect, the optimal repetition rates give the pilots very little excess experience. Note that the contribution from mission $j$ to a competency $i$ is either equal to the total number of sorties $T_j$ or equal to 0, for all competencies $i$. Therefore, the bars in figure 6.4 that have the same color all have the same height.

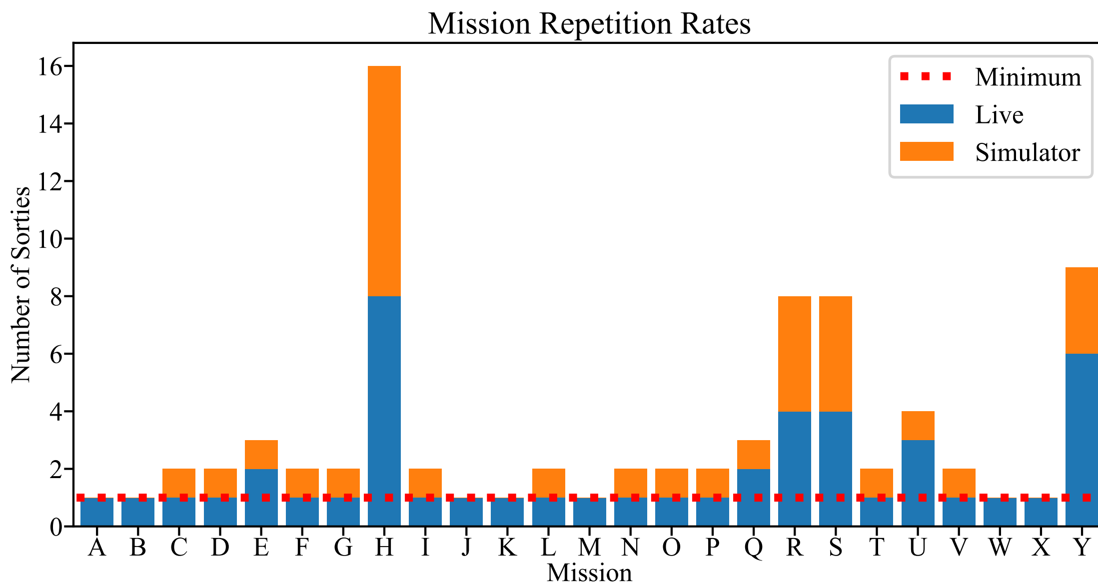



*Figure 6.4 Repetition rates for the baseline problem. The number of sorties for each mission live $L_j$ and in the simulator $S_j$ is shown in blue and orange, respectively. The minimum number of live sorties for each mission $K_j$ is indicated with the red dotted lines.*

## 6.3 Focus-Competency Problem

When we include the limit for the number of competencies practiced in each sortie, the number of possible states to consider increases drastically. Nevertheless, we are able to find optimal repetition rates using the IP3-implementation of section 4.3.2. In this section, we illustrate one such solution in the form of bar charts similar to those of section 6.2.

We use the same parameters as for the baseline problem, see section 6.1, but we add the requirement that no more than 3 competencies are practiced in any of the sorties. Compared to the baseline optimization problem, the number of competencies practiced in each sortie is reduced for most missions. Therefore, the required number of sorties is significantly higher in the focus-competency optimization problem. The optimal repetition rates give a total of 58 live sorties and 44 sorties in the simulator. In order to reduce the search space and make the training program as varied as possible, we set the maximum number of repetitions $\tilde{Q}$ for each combination of competencies to 7. This is the smallest value of $\tilde{Q}$ that we can use without increasing the total cost of the training program.

For the combination-of-competencies optimization problem, each set of repetition rates is specified by the numbers of live- and simulator sorties for each combination $L_{j,k}$ and $S_{j,k}$. The same training programs occur in the individual-sortie optimization problem, but it is represented in another way. The optimal repetition rates are illustrated in figure 6.6. Note that for all missions except R and S, no more that two combinations of competencies are actually carried out, even though there are at least 4 possible combinations for most missions. As for the baseline problem, the four missions H, R, S, and Y make up a significant portion of the training program.

Figure 6.7 demonstrates that the pilots get the required amount of practice for each competency. As for the baseline problem (see figure 6.5) the pilots get very little excess experience beyond the

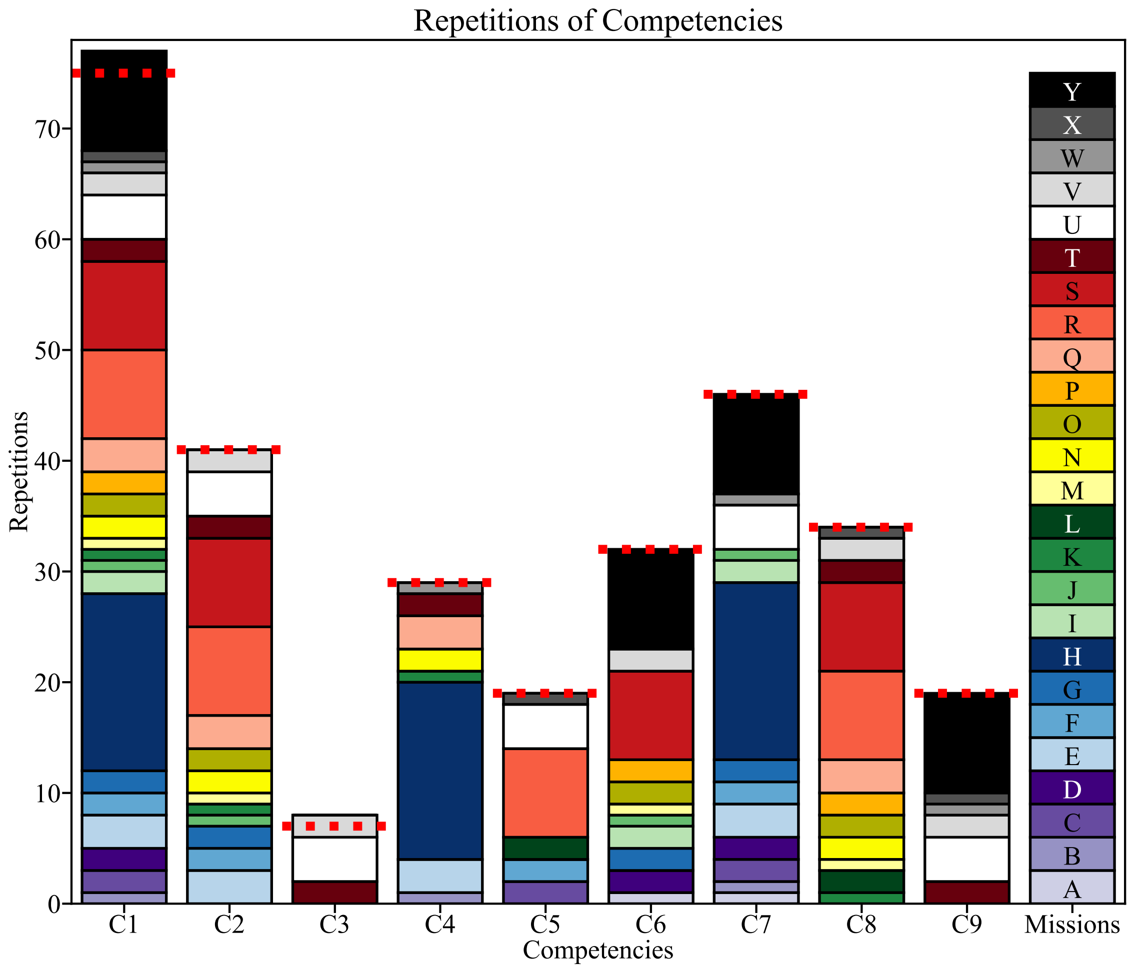



*Figure 6.5 Number of times the pilots practice each competency for the repetition rates of figure 6.4, illustrated with colored bars. The figure demonstrates that these repetition rates fulfill the requirements for the number of repetitions for each competency $B_i$ in the baseline optimization problem. The competency requirements $B_i$ are indicated by the red dotted lines. The colors of the bars indicate which mission gave rise to each contribution.*

requirements. Each bar represents a combination of a competency $i$ and a mission $j$, and the height of the bar is the sum of the number of sorties $T_{j,k}$ for all combinations $k$ that contain competency $i$.

## 6.4 Computation Time

The main objective of this work is to explore the available methods for solving the COMFORT optimization problem. The different solvers in OR-Tools and the two variants of the focus-competency optimization problem represent the different methods that we compare. In the comparison, we focus on the computation time used by the solver and the total cost of the resulting training programs. Given enough time, all the implementations would return an optimal solution. However, since we are able to find solutions relatively quickly, we limit the computation time to a minute and compare

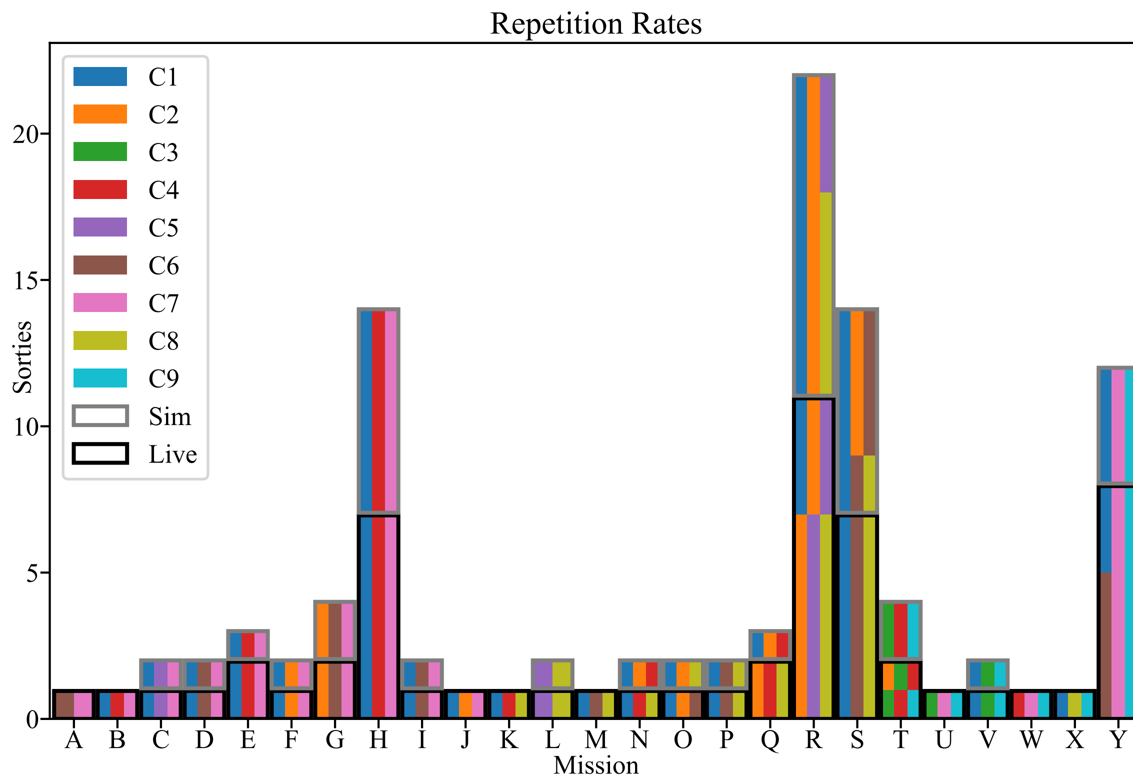



*Figure 6.6 Optimal repetition rates for the focus-competency optimization problem. The black outlines represent the number of live sorties $L_j$ for each mission $j$, and the gray outlines indicate the number of simulator sorties $S_j$. These outlines can be compared to the repetition rates for the baseline problem in figure 6.4. Within the outlines, there are colored bars representing the competencies, C1 to C9, that were practiced in each sortie.*

the best solutions found in that time frame.

| Implementation | Computation Time | Status | Cost |
|---|---|---|---|
| IP1 | 0.246 s | Optimal | 32.378 |
| SAT1 | 1.721 s | Optimal | 32.362 |
| CP1 | 56 s | Feasible | 38.946 |

*Table 6.1 Comparison of the three variant implementations of the baseline optimization problem. The table contains the computation time used to solve the example problem of section 6.1, status returned by the solver, and total cost of the best solutions found. The IP1- and SAT1- variant implementations return optimal solutions, whereas for CP1 we list the best solution found within a minute of computation time.*

When comparing the solvers, we focus on the baseline problem, since the implementations CP1, SAT1, and IP1 are very analogous. The computation times used to solve the example problem and

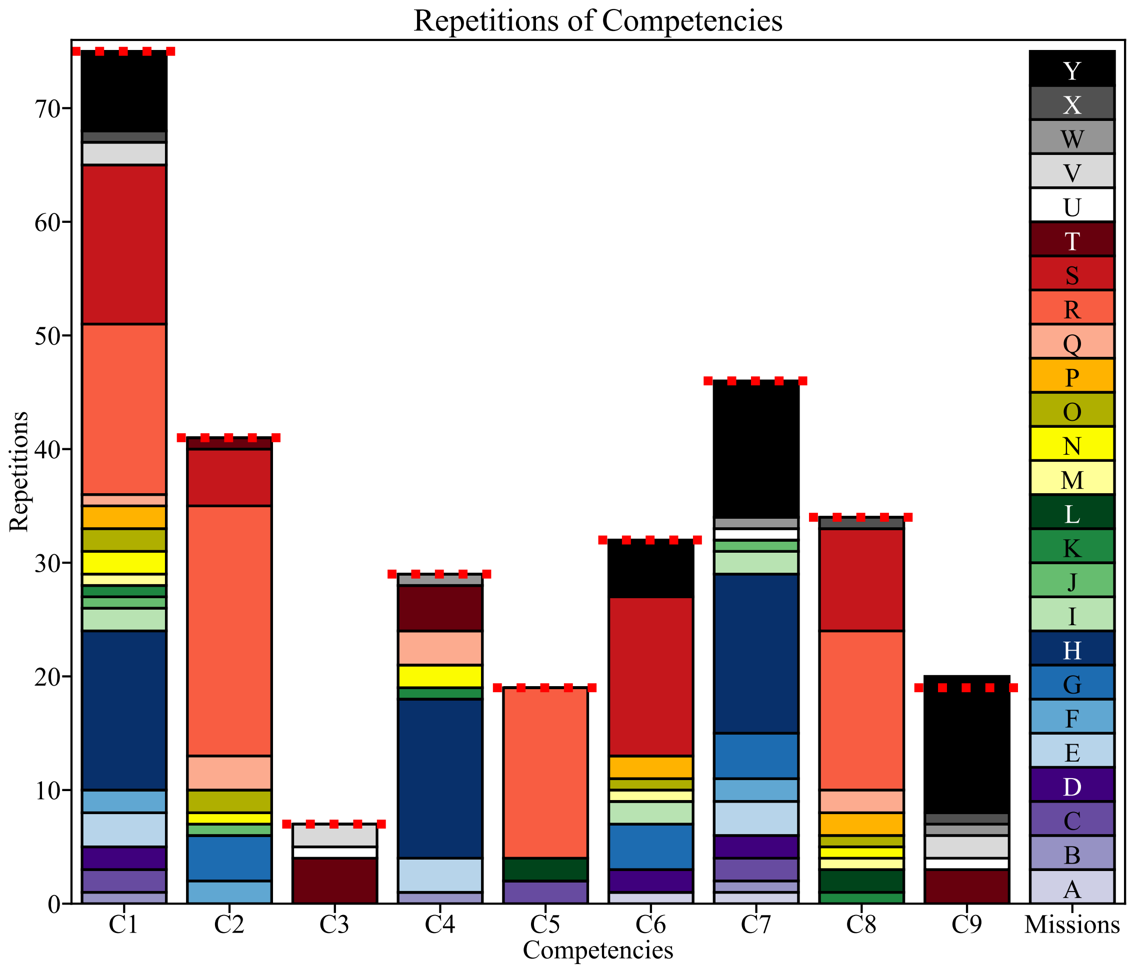



*Figure 6.7 Number of times the pilots practice each competency for the repetition rates of figure 6.6, for the focus-competency optimization problem, illustrated with colored bars. The figure demonstrates that the repetition rates fulfill the requirements for the number of repetitions for each competency $B_i$, which are indicated by the red dotted lines. The colors of the bars indicate which mission gave rise to each contribution.*

the total costs of the solutions are summarized in table 6.1. The upper bound $Q$ for the variables $L_j$ and $S_j$ was equal to 10 in all cases. Both the SAT solver and the CBC IP solver return optimal solutions relatively quickly. However, the more specialized IP solver algorithm is faster by a factor of around 7. The total costs returned by the SAT- and CBC solvers differ slightly due to rounding errors caused by converting the costs into integers for the SAT solver. This conversion is performed by multiplying with $V = 1000$ and rounding down, see section 5.1.

The best repetition rates found by the CP1 variant implementation are shown in figure 6.8. These rates are very different from the optimal rates of figure 6.4. The constraint solver has only added additional sorties beyond the minimum requirement $K_j$ for the last half of the missions. This indicates that only a very small fraction of the possible states have been evaluated, even after almost a minute of computation time. Therefore, the computation time required to ensure that the optimal

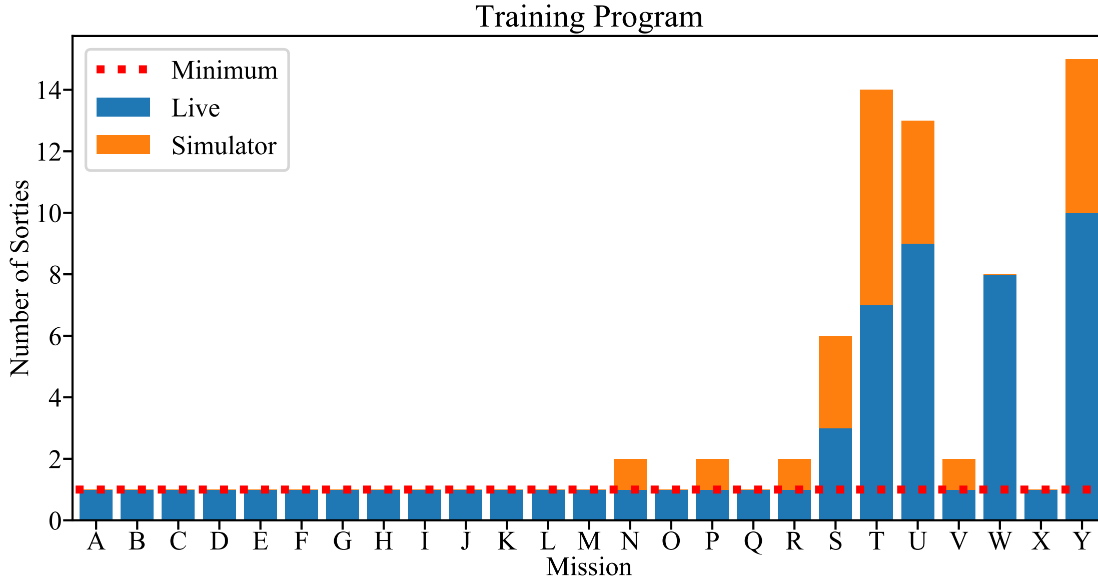



*Figure 6.8* *Non-optimal training program for the baseline version of the optimization problem, calculated with the CP-solver. The number of sorties for each mission live $L_j$ and in the simulator $S_j$ is shown in blue and orange, respectively. The minimum number of live sorties $K_j$ for each mission is indicated with the red dotted lines.*

repetition rates are found may be several orders of magnitude longer than one minute. The constraint solver has also been unable to maximize the ratio of simulator sorties to live sorties. This indicates that the configuration of the *DecisionBuilder* in the CP1 variant implementation is inefficient.

For the focus-competency optimization problem, the IP3 variant implementation quickly returns an optimal solution. The computation time used by the IP3 variant implementation and the total cost of the optimal solution is given in table 6.2. IP3 solves the focus-competency problem even faster than IP1 solves the baseline problem, although these results are not directly comparable. For IP3, we did not include the computation time needed to find the combinations of competencies, and the upper bound for each variable $\tilde{Q}$ was equal to 7, whereas $Q = 10$ for IP1.

| Implementation | Computation Time | Status | Cost |
|---|---|---|---|
| IP3 | 0.132 s | Optimal | 38.813 |
| CP2 | $\geq$ 60 s | No Solution Found | - |

*Table 6.2* *Comparison of the CP2- and IP3 variant implementations of the focus-competency optimization problem, for the example problem of section 6.1. The table contains the computation time used, the status of the solver, and the total cost of the best solutions found. IP3 returns an optimal solution, whereas the CP2 does not find any feasible solutions.*

By contrast, the CP2 variant implementation is unable to find a feasible solution, even if we allow the solver to run for several minutes. Both the variant of the optimization problem and the algorithm of the solver differs between CP2 and IP3. Therefore, these results are not well suited

to compare either the two optimization problems or the two solvers. However, it is clear that IP3 represents an effective approach to the focus-competency optimization problem.

# 7 Discussion

Competency-based training is an approach where training is planned according to a set of competency requirements. COmpetency-Mission Frequency Optimizer for Readiness Training (COMFORT) determines optimal mission repetition rates for a simplified competency-based training regime. Competency-based training has been adopted in a wide-range of fields, and may bring about big changes to the field of pilot training. However, such a large transition will take several years, and many aspects of the final training regime are still undetermined. Therefore, we focus on the available methods for the optimization, rather than a specific case. COMFORT exemplifies how constraint programming may be applied for planning purposes. COMFORT does not involve simulations or detailed planning of the training process, but may be useful for planning or scheduling purposes.

The optimization of the repetition rates is formulated as a constrained optimization problem (COP). We define three different variants of the COP, implement solution algorithms for all three variants, and discuss the differences in complexity between the variants. In order to demonstrate COMFORT, we define a set of example parameters that we use together with all three variants of the COP. We also consider three different algorithms for solving COPs: two general algorithms and one which is specialized to integer programs (IPs). We compare the computation time required to solve the example problem for all three variants of the COP and for all three COP solution algorithms.

## 7.1 Solution Algorithms

The software package OR-Tools developed by Google [33] provides implementations of all three COP solution algorithms that we consider. The three implementations are referred to as the constraint solver, the SAT solver, and the CBC solver, see chapter 3. In addition to the computation time, we compare the available functionality of the three solvers.

The constraint solver is a general-purpose CSP solver, and contains the most extensive functionality. For instance, none of the other solvers provide an equivalent to the *makeDistribute*-method used in section 5.4. However, for our implementations of COMFORT it is too slow to be practical, see section 6.4. We note that there are many possible ways that one could optimize the search and simplify the problem, but we have not spent much time on such optimizations. In our estimation, spending our time optimizing the other implementations is likely to yield better results.

The SAT solver is intended to replace the constraint solver in OR-Tools as a general-purpose CSP solver [33]. The developers claim that it is faster than the constraint solver for most practical cases [33], and our implementation is indeed faster with the constraint solver by multiple orders of magnitude, see section 6.4. For the baseline COMFORT problem, it finds an optimal solution in less than 2 seconds. Clearly, the SAT solver can provide great performance improvements compared to the constraint solver, but we cannot conclude from our single example that this will hold for other cases.

Finally, we showed that the COMFORT problem can be formulated as an integer program, for which there exists specialized solvers that can be much faster than general-purpose CSP solvers. We implemented COMFORT using the IP solver class of OR-Tools, and solved the problem using the open-source solver CBC [36]. As expected, this implementation turned out to be several times faster than the SAT solver for the baseline COMFORT problem, and it enabled us to solve the more complex focus-competency optimization problem quickly. There are also alternative IP solvers that claim to be even faster than CBC [39]. Since the IP solvers are very efficient, it is useful to determine if a CSP has this form, or if it can be converted to this form in a simple way.

## 7.2 Complexity

The number of variables in a COP and the sizes of their domains can affect the computation time needed to solve the COP strongly. Therefore, we use these two properties as a measure of the complexity of a COP. However, the dependency can vary between different COPs and different solution algorithms. For example, the domain size does not necessarily affect the Branch-and-Bound algorithm at all since it focuses on the constraints, see section 2.2.2. By contrast, for the constraint solver, the computation time often scales linearly with the domain size of a variable.

The three different variants of the COMFORT constraint optimization problem have different sets of variables, and therefore different complexities. We focus on the baseline optimization problem and the combinations-of-competencies variant of the focus-competency problem, since these are the two COPs that we were able so solve so far.

For the baseline problem, the number of variables is simply given by the number of missions. However, for the combination-of-competencies COP, the number of variables is given by the total number of possible combinations of competencies. The number of such combinations ($Y_j$) is determined by the number of competencies that are possible to practice in a mission ($H_j$) and the number of competencies that can be practiced in a single sortie ($D_j$). In many cases, the number of possible combinations ($Y_j$) will increase drastically if we add one additional possible competency to the mission ($H_j$ increases by 1), see equation 4.9.

The domain sizes are given by the maximum number of repetitions for each mission ($Q$) or combination of competencies ($\tilde{Q}$). In order to reduce complexity, we want to choose the lowest possible values of these upper limits. However, the allowed number of repetitions must be high enough that all the competency requirements can be satisfied. In order to find optimal repetition rates, we should also allow for extra repetitions of the most cost-effective missions and combinations. The required number of repetitions of a competency typically reflects a frequency requirement. Therefore, we can reduce the domain sizes by reducing the length of each planning period.

## 7.3 Improvements

There are many possible changes to the COMFORT model and implementation that could allow us to solve the example problem faster. However, such optimizations will not necessarily work for other problems. For example, in the current version, the number of repetitions of a mission live and in the simulator are represented by two separate variables ($L_j$ and $S_j$). For missions where the maximum allowed ratio of simulator and live training is a simple one such as 1 to 1, 1 to 0, or 0 to 1, one can replace the two variables ($L_j$ and $S_j$) with a single one. If we are unable to solve an optimization problem, we can also modify the problem to reduce complexity, although this may change the end results. For example, for missions where the number of possible combinations of competencies is high, we can remove one of the allowed competencies. In that case, one should try to remove competencies that are easy to practice in other missions. More drastically, we can split a long period of training into shorter ones with smaller sets of missions. This can give a reduction in both the number of variables and the size of each domain.

The COMFORT model can be extended in several ways in order to better reflect an actual training regime. For instance, complexity factors representing operating conditions that affect the pilots may be taken into account. As a first approach to include complexity factors, one can create additional missions where the same tasks are carried out under different conditions and add additional competencies related to mastering the new operating conditions. Additionally, one may

want to create different training programs for groups of pilots in the same squadron with different levels of experience and qualifications. Then, the COMFORT model should be extended in order to ensure that the resulting training programs are compatible.

## 7.4 Conclusion

As we discussed in the introduction, efficient pilot training is important for the RNoAF, and competency-based training is a possible approach to improve efficiency. COMFORT demonstrates how COP solution algorithms can be used for planning or scheduling purposes in competency-based training regimes. We focus on the available methods and algorithms, rather than the details of the COP solutions. We found that the new SAT solver from OR-Tools can provide great performance benefits compared to the constraint solver, although we only looked at a single COP in the comparison. We also showed that specialized solvers for integer programs can be even more efficient, and discussed how IPs differ from general COPs.

# References

[1] M. Hafsten, "Implikasjoner av innføringen av et nettverksbasert simulatoranlegg i kampflyvåpenet," Master's thesis, The Norwegian Defence University College, 2016, (BEGRENSET).

[2] Forsvarsdepartementet, "Konseptuell løsning for prosjekt 7600 Fremtidig kampflykapasitet," 2006.

[3] Forsvarsdepartementet, "Nye kampfly til forsvaret," 2008.

[4] Forsvarsdepartementet, "Et forsvar til vern om Norges sikkerhet, interesser og verdier," 2008.

[5] G. Skogsrud and O. M. Mevassvik, "Simulation of a combat training program for Norwegian pilots," in *RTO meeting proceedings of the NATO Modeling and Simulation Group*, Bern, Switzerland, Oct. 2011.

[6] G. Skogsrud and O. M. Mevassvik, "TREFF - modellbeskrivelse," Forsvarets forskningsinstitutt, FFI-rapport 2012/01240 (Begrenset), 2012.

[7] G. Skogsrud, "TREFF - implementering og brukerveiledning," Forsvarets forskningsinstitutt, FFI-rapport 2012/01237 (Unntatt offentlighet), 2012.

[8] G. K. Svendsen and O. M. Mevassvik, "Konsekvenser av værkansellering for trening av F-35 flygere," Forsvarets forskningsinstitutt, FFI-notat 2014/00474 (Begrenset), 2014.

[9] O. M. Mevassvik *et al.*, "Innspill til valg av simulator for F-35," Forsvarets forskningsinstitutt, FFI-rapport 2013/00934 (Begrenset), 2013.

[10] G. K. Svendsen and O. M. Mevassvik, "Periodisering av årsplan for F-35 trening," Forsvarets forskningsinstitutt, FFI-rapport 2015/00449 (Begrenset), 2015.

[11] G. K. Svendsen *et al.*, "Optimized pilot training for combat aircraft," in *Interservice/Industry Training, Simulation and Education Conference (I/ITSEC) 2017*, no. 17199, Orlando, FL, Dec. 2017.

[12] G. K. Svendsen and O. M. Mevassvik, "Føringsbasert optimering for videreutvikling av TREFF," Forsvarets forskningsinstitutt, FFI-rapport 2014/00473 (Begrenset), 2014.

[13] I. U. Haugstuen and G. K. Svendsen, "Sentrale faktorer for trening med F-35," Forsvarets forskningsinstitutt, FFI-rapport 17/16681 (Begrenset), 2017.

[14] Forsvarsdepartementet, "F-35 konsept for simulatorer og tekniske trenere," 2013, (Unntatt offentlighet).

[15] Luftforsvaret, "Provisions for operational classification of F-16 aircrew BFL 110-10 (F)," 2014, (RESTRICTED // REL TO NOR and USA).

[16] Luftforsvaret, "Regulations for combat ready training of F-16 aircrew RFL 110-20 (A)," 2014, (RESTRICTED // REL TO NOR and USA).

[17] Luftforsvaret, "Grunnlag for F-35 treningsprogram for kampklare flygere," 2011, (BEGRENSET).

[18] W. C. Rivenbark and W. S. Jacobson, "Three principles of competency-based learning: Mission, mission, mission," *Journal of Public Affairs Education*, vol. 20, no. 2, pp. 181–192, 2014.

[19] D. L. Roberson and M. C. Stafford, *The Redesigned Air Force Continuum of Learning*. Air University Press, 2017.

[20] P. Crane *et al.*, "Advancing fighter employment tactics in the Swedish and US air forces using simulation environments," in *RTO meeting proceedings of the NATO Modeling and Simulation Group*, Neuilly-sur-Seine, France, Oct. 2006.

[21] C. M. Colegrove and W. Bennett Jr., "Competency-based training: Adapting to warfighter needs," Air Force Research Laboratory, AFRL-HE-AZ-TR-2006-0014, 2006.

[22] B. T. Schreiber, M. Schroeder, and W. Bennett Jr., "Distributed mission operations within-simulator training effectiveness," *The International Journal of Aviation Psychology*, vol. 21, no. 3, pp. 254–268, 2011.

[23] W. Arthur Jr. *et al.*, "Factors that influence skill decay and retention: A quantitative review and analysis," *Human Performance*, vol. 11, no. 1, pp. 57–101, 1998.

[24] J. van der Pal and H. Abma, "Competency-based pilot training for the RNLAF," in *International Training and Education Conference (ITEC)*, Brussels, May 2009.

[25] J. van der Pal, E. Boland, and M. de Rivecourt, "Competency-based design of F-16 qualification training," in *Proceedings of the 2009 Interservice/Industry Training, Simulation and Education Conference (I/ITSEC-2009)*, 2009.

[26] E. Tsang, *Foundations of constraint satisfaction*. Academic Press, 1993.

[27] R. Hemmecke *et al.*, "Nonlinear integer programming," in *50 Years of Integer Programming 1958-2008*. Springer, 2010, pp. 561–618.

[28] G. Dantzig, *Linear Programming and Extensions*. Princeton University Press, 2016.

[29] S. P. Bradley, A. C. Hax, and T. L. Magnanti, *Applied Mathematical Programming*. Addison-Wesley, 1977.

[30] M. Padberg and G. Rinaldi, "A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems," *SIAM review*, vol. 33, no. 1, pp. 60–100, 1991.

[31] A. M. Law, *Simulation Modeling & Analysis*, 4th ed. McGraw-Hill, 2007, International Edition.

[32] Anylogic. The AnyLogic Company. [Online]. Available: http://www.anylogic.com

[33] Google OR-Tools. Google. [Online]. Available: http://code.google.com/p/or-tools/

[34] J. P. M. Silva and K. A. Sakallah, "Grasp—a new search algorithm for satisfiability," in *Proceedings of International Conference on Computer Aided Design*, 1996, pp. 220–227.

[35] R. J. Bayardo Jr. and R. Schrag, "Using CSP look-back techniques to solve real-world SAT instances," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 1997, pp. 203–208.

[36] Cbc. Coin-OR Foundation. [Online]. Available: http://projects.coin-or.com/CBC

[37] Glpk. Free Software Foundation. [Online]. Available: http://www.gnu.org/software/glpk

[38] Scip. Zuse Institute Berlin. [Online]. Available: http://scip.zib.de

[39] Gurobi. Gurobi Optimization, LLC. [Online]. Available: http://www.gurobi.com

[40] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, Eds. Springer US, 1972, pp. 85–103.

[41] I. P. Gent, K. E. Petrie, and J.-F. Puget, "Chapter 10 - symmetry in constraint programming," in *Handbook of Constraint Programming*, ser. Foundations of Artificial Intelligence, F. Rossi, P. van Beek, and T. Walsh, Eds. Elsevier, 2006, vol. 2, pp. 329 – 376.

[42] Luftforsvaret, "Annual Training Program (ATP) for F-35 pilots, RFL 115-20 (B)," 2018, (RESTRICTED // REL TO NOR and USA).

# About FFI

The Norwegian Defence Research Establishment (FFI) was founded 11th of April 1946. It is organised as an administrative agency subordinate to the Ministry of Defence.

## FFI's MISSION

FFI is the prime institution responsible for defence related research in Norway. Its principal mission is to carry out research and development to meet the requirements of the Armed Forces. FFI has the role of chief adviser to the political and military leadership. In particular, the institute shall focus on aspects of the development in science and technology that can influence our security policy or defence planning.

## FFI's VISION

FFI turns knowledge and ideas into an efficient defence.

## FFI's CHARACTERISTICS

Creative, daring, broad-minded and responsible.

# Om FFI

Forsvarets forskningsinstitutt ble etablert 11. april 1946. Instituttet er organisert som et forvaltningsorgan med særskilte fullmakter underlagt Forsvarsdepartementet.

## FFIs FORMÅL

Forsvarets forskningsinstitutt er Forsvarets sentrale forskningsinstitusjon og har som formål å drive forskning og utvikling for Forsvarets behov. Videre er FFI rådgiver overfor Forsvarets strategiske ledelse. Spesielt skal instituttet følge opp trekk ved vitenskapelig og militærteknisk utvikling som kan påvirke forutsetningene for sikkerhetspolitikken eller forsvarsplanleggingen.

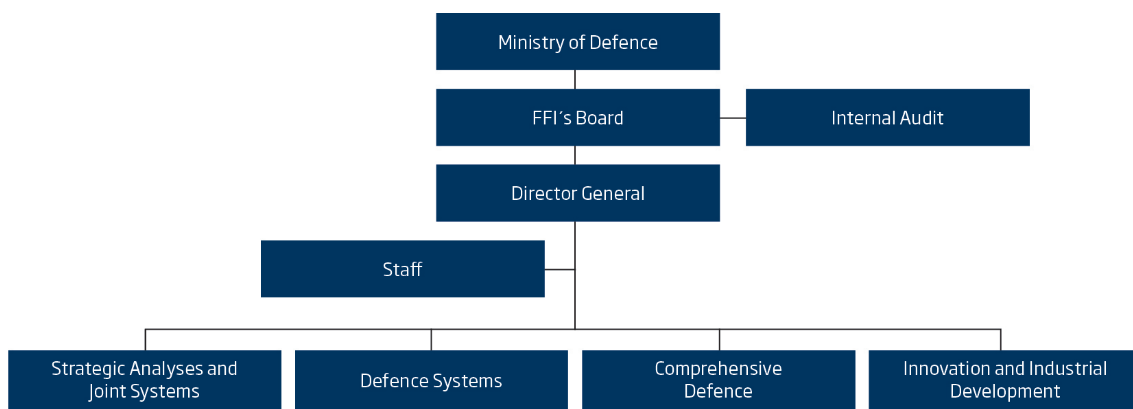
## FFIs VISJON

FFI gjør kunnskap og ideer til et effektivt forsvar.

## FFIs VERDIER

Skapende, drivende, vidsynt og ansvarlig.

# FFI's organisation



**Forsvarets forskningsinstitutt**
Postboks 25
2027 Kjeller

Besøksadresse:
Instituttveien 20
2007 Kjeller

Telefon: 63 80 70 00
Telefaks: 63 80 71 15
Epost: ffi@ffi.no

**Norwegian Defence Research Establishment (FFI)**
P.O. Box 25
NO-2027 Kjeller

Office address:
Instituttveien 20
N-2007 Kjeller

Telephone: +47 63 80 70 00
Telefax: +47 63 80 71 15
Email: ffi@ffi.no

FFI Forsvarets forskningsinstitutt
Norwegian Defence Research Establishment