

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Intelligent Planning Optimization for Programmed Stops at CERN

Manuel Carlos Ramos Alves



Mestrado em Engenharia Informática e Computação

Supervisor: Prof. Daniel Castro Silva

Second Supervisor: Prof. Alexandre Valle de Carvalho

Supervisor at CERN: Fernando Baltasar dos Santos Pedrosa

July 20, 2026

Intelligent Planning Optimization for Programmed Stops at CERN

Manuel Carlos Ramos Alves

Mestrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

President: Prof. Konstantinos Bletsas

Examiner: Prof. Pedro Abreu

Supervisor: Prof. Daniel Castro Silva

July 20, 2026

Resumo

O planeamento de paragens programadas no CERN, como as paragens técnicas de fim de ano durante as quais os seus aceleradores são mantidos e melhorados, exige coordenar várias centenas de tarefas interdependentes sob fortes restrições temporais e de recursos. Como cada dia adicional de paragem é dispendioso, a qualidade desta ordenação importa diretamente; no entanto, continua a ser feita em grande parte à mão no Microsoft Project, uma ferramenta que regista um escalonamento mas pouco ajuda a raciocinar sobre como as suas muitas restrições interagem. As ferramentas atuais, além disso, apoiam ou a inspeção visual de um escalonamento ou a sua otimização, raramente ambas sobre uma única abstração reutilizável.

Esta dissertação propõe e avalia um fluxo de trabalho integrado e centrado num *template*, para as intervenções de manutenção recorrentes que constituem grande parte de uma paragem. A abstração central é o *template*: uma entidade parametrizada e reutilizável que é, a um tempo, a representação editável que o coordenador tem de um plano recorrente e a entrada estruturada de um otimizador de programação por restrições, que minimiza a duração da paragem conciliando tempo útil e tempo de calendário numa escala comum. Em torno desta abstração, uma interface interativa expõe o escalonamento, as suas restrições e as alterações que o otimizador propõe, para que o coordenador as possa inspecionar, corrigir e confirmar; o fluxo fecha assim o ciclo, da autoria de um plano uma só vez até uma exportação revista de volta para o Microsoft Project, mantendo o humano no controlo do resultado.

O protótipo, construído como uma extensão da plataforma Scheduling Tools do CERN, foi avaliado em três vertentes e um estudo de caso real: um estudo de usabilidade com vinte membros do CERN, um questionário comparativo com coordenadores de planeamento e um *benchmark* algorítmico controlado do otimizador. O estudo de caso reconstruiu uma paragem real de fim de ano do Proton Synchrotron a partir de *templates* reutilizáveis.

O protótipo obteve 81,5 na System Usability Scale, um resultado independente da experiência prévia em planeamento, e os coordenadores consideraram o fluxo uma melhoria face à prática atual. O *benchmark* mostrou que o otimizador devolve escalonamentos comprovadamente ótimos à dimensão e à estrutura de dependências de uma paragem real. No estudo de caso, seis *templates* cobriram 74% da paragem e o otimizador reproduziu a sua duração de forma exata, em segundos, deixando apenas a ordenação residual das tarefas para o conhecimento tácito do coordenador. A afirmação central não é, portanto, um novo algoritmo de escalonamento, mas uma abordagem integrada e centrada num *template*, concretizada e validada para as paragens programadas no CERN.

Palavras-chave: Escalonamento de Projetos com Recursos Limitados, Programação por Restrições, Visualização de Escalonamentos, Gráfico de Gantt Interativo

Abstract

The planning of programmed stops at CERN, such as the year-end technical stops during which its accelerators are maintained and upgraded, requires coordinating several hundred interdependent tasks under tight temporal and resource constraints. Because every additional day of stoppage is costly, the quality of this sequencing matters directly, yet it is still carried out largely by hand in Microsoft Project, a tool that records a schedule but offers little help in reasoning about how its many constraints interact. Existing tools, moreover, support either the visual inspection of a schedule or its optimization, rarely both over a single reusable abstraction.

This dissertation proposes and evaluates an integrated, template-centric workflow for the recurring maintenance interventions that make up much of a stop. The central abstraction is the template: a parameterized, reusable entity that is at once the coordinator's editable representation of a recurring plan and the structured input to a constraint-programming optimizer, which minimizes the stoppage duration while reconciling working time and calendar time on a common scale. Around this abstraction, an interactive interface exposes the schedule, its constraints, and the changes the optimizer proposes, so that the coordinator can inspect, correct, and commit them; the workflow thus closes the loop from authoring a plan once to a reviewed export back to Microsoft Project, keeping the human in control of the result.

The prototype, built as an extension of the CERN Scheduling Tools platform, was evaluated along three tracks and a real case study: a usability study with twenty CERN members, a comparative questionnaire with planning coordinators, and a controlled algorithmic benchmark of the optimizer. The case study reconstructed a real Proton Synchrotron year-end stop from reusable templates.

The prototype scored 81.5 on the System Usability Scale, a result independent of prior planning experience, and coordinators judged the workflow an improvement over their current practice. The benchmark showed that the optimizer returns proven-optimal schedules at the size and dependency structure of a real stop. In the case study, six templates covered 74% of the stop and the optimizer reproduced its duration exactly, in seconds, leaving only the residual task ordering to the coordinator's tacit expertise. The central claim is therefore not a new scheduling algorithm but an integrated, template-centric approach, realized and validated for the programmed stops at CERN.

Keywords: Resource-Constrained Project Scheduling, Constraint Programming, Schedule Visualization, Interactive Gantt Chart

UN Sustainable Development Goals

The United Nations Sustainable Development Goals (SDGs) provide a global framework to achieve a better and more sustainable future for all. It includes 17 goals to address the world's most pressing challenges, including poverty, inequality, climate change, environmental degradation, peace, and justice.

This dissertation develops an integrated, template-centric approach for planning the programmed maintenance stops of CERN's particle accelerators. Examined in the light of the SDGs, its impact falls most directly under two goals: **SDG 9**, by improving the efficiency, reliability, and tooling of a major scientific research infrastructure, and **SDG 8**, by raising the productivity and quality of the planning work itself. Its relation to other goals, such as the facility's energy footprint, is at most indirect and is not claimed here.

The specific Sustainable Development Goals mentioned have the following names:

SDG 9 Build resilient infrastructure, promote inclusive and sustainable industrialization and foster innovation

SDG 8 Promote sustained, inclusive and sustainable economic growth, full and productive employment and decent work for all

SDG	Target	Contribution	Performance Indicators and Metrics
9	9.1	Optimized planning of programmed stops helps keep the accelerator infrastructure reliable and shortens its maintenance downtime.	Stop makespan; proven-optimal schedules at real-stop scale; 74% coverage from reusable templates.
	9.5	A research and engineering innovation — constraint-programming optimization with a human-in-the-loop visual interface — that modernizes the planning tooling of a leading scientific infrastructure.	Coordinator adoption: a System Usability Scale of 81.5; all seven surveyed coordinators willing to adopt.
8	8.2	The approach raises the productivity of planning coordinators, replacing manual, error-prone scheduling with reusable templates and an optimizer, so that effort shifts to expert judgment.	Usability independent of planning background (SUS 81.5); reduced task time, errors, and assists; perceived improvement over current practice.

Acknowledgements

Aos meus orientadores, o Professor Daniel Castro Silva e o Professor Alexandre Valle de Carvalho, obrigado por terem abraçado este projeto comigo, pelo rigor e pelo cuidado com que me apontaram sempre o caminho, e por estarem disponíveis em cada passo.

Um obrigado muito especial ao Fernando Pedrosa, que me abriu as portas do CERN e me acompanhou ao longo de toda esta aventura. E a todas as pessoas que ali conheci e que, de mil maneiras diferentes, me ajudaram e me fizeram sentir parte.

À minha família e amigos, devo tudo o que sou. Estiveram sempre presentes, em cada decisão e em cada dia. Nada disto faria sentido sem vocês.

E aos meus "primos(as)": obrigado pela companhia que não conhece limites. Espalhados pelos quatro cantos do mundo, com loucuras pelo meio, são vocês a casa a que volto sempre, esteja eu onde estiver.

Manuel Carlos Ramos Alves

Declaration of honour

I declare that this dissertation is my own work and has not previously been submitted or assessed at this or any other institution. References to other authors (statements, ideas, thoughts), as well as the use of published works, strictly comply with the rules of attribution and are duly identified in the text and in the bibliographic references, in accordance with the applicable referencing standards. I am aware that the practice of plagiarism or self-plagiarism constitutes an academic offence, as established in the U.Porto Code of Ethics.

I also declare that I have used Artificial Intelligence tools to complete part(s) of this dissertation, and that this use and its results have been carefully checked for errors, inaccuracies, unfounded attributions or other forms of bias in content and arguments, as well as determining authorship.

Manuel Carlos Ramos Alves

Manuel Alves

“Nobody who ever gave his best regretted it”

George Halas

Contents

1	Introduction	1
1.1	Context	1
1.2	Motivation	2
1.3	Problem	3
1.4	Objectives and Contributions	3
1.5	Document Structure	4
2	Background	6
2.1	Scheduling Foundations	6
2.1.1	Scheduling Contexts and Problem Types	6
2.1.2	Core Concepts and Representations	7
2.1.3	Constraints and Objectives	8
2.2	CERN Planning Context	9
2.2.1	Programmed Stops	10
2.2.2	Planning Workflow and Data	11
2.3	Visualization Foundations	14
2.3.1	Plan Visualization Requirements	14
2.3.2	Interaction Concepts	16
2.4	Summary	16
3	Related Work	17
3.1	Literature Review Process	17
3.1.1	Search Strategy	17
3.1.2	Identification Phase	18
3.1.3	Screening Phase	19
3.1.4	Review and Analysis	20
3.2	Visualization	22
3.2.1	Time-Oriented Visualizations	23
3.2.2	Dependency and Constraint Visualizations	26
3.2.3	Spatiotemporal and Spatial Visualizations	28
3.2.4	Visual Analytics and Interactive Planning	31
3.2.5	Constraint Specification Interfaces	34
3.2.6	Comparison of Approaches	36
3.3	Optimization	37
3.3.1	Exact Methods	38
3.3.2	Heuristics and Metaheuristics	41
3.3.3	Multi-Criteria and Dynamic Scheduling	45
3.3.4	Comparison of Methods	47

3.4	Summary	49
4	Solution Specification	50
4.1	Approach Overview	51
4.2	Problem and Requirements	51
4.2.1	The Planning Problem	52
4.2.2	Scope of the Planning Problem	52
4.2.3	Requirements	53
4.3	The Template Abstraction	53
4.3.1	Absolute Dates versus Relative Offsets	55
4.3.2	Parameterized Templates	56
4.4	Conceptual Architecture	57
4.5	The Scheduling Problem	58
4.5.1	Tasks and Time	59
4.5.2	Constraints	60
4.5.3	Objective	61
4.5.4	Explaining Infeasibility	61
4.6	Visualization Design	61
4.7	The Review Workflow	63
4.8	Summary	64
5	Implementation	65
5.1	Technology Stack and Deployment	65
5.2	The Template Engine	66
5.2.1	Storage	66
5.2.2	Capturing Templates from Existing Plans	66
5.2.3	Instantiation by Temporal Shift	67
5.2.4	Batch Instantiation	68
5.2.5	Implementing Parameterized Templates	68
5.2.6	The Template Lifecycle	70
5.3	The Optimizer	71
5.3.1	The Dual-Timeline Encoding	71
5.3.2	Anchor-Relative Constraint Dates	72
5.3.3	Encoding the Concurrency Limits	73
5.3.4	Search Strategy and Solver Parameters	74
5.3.5	The Two-Phase Warm Start	75
5.3.6	Infeasibility Diagnostics	75
5.4	The Interactive Interface	76
5.4.1	Persisting and Adjudicating Runs	78
5.4.2	Export to Microsoft Project	78
5.5	Challenges	81
5.6	Summary	82
6	Evaluation Protocol	83
6.1	Objectives and Research Questions	83
6.2	Evaluation Design	84
6.2.1	Track A: Features and Usability	84
6.2.2	Track B: Comparative Workflow Questionnaire	85
6.2.3	Track C: Algorithmic Benchmark	86

6.3	Participants, Ethics, and Moderation	87
6.4	Apparatus and Instruments	87
6.5	Task Scenarios and Metrics	88
6.5.1	Task Ratings	90
6.5.2	Performance Metrics (Objective)	90
6.5.3	Perception Metrics (Subjective)	90
6.5.4	Algorithmic Metrics (Track C)	91
6.5.5	Mapping Tasks to Research Questions	92
6.5.6	Data Analysis Plan	92
6.6	Pilot Study	93
6.7	Summary	93
7	Results	95
7.1	Sample Characterization	95
7.2	Track A: Usability Results	96
7.2.1	Task Performance	96
7.2.2	Perceived Difficulty and Usefulness	98
7.2.3	System Usability Scale	99
7.2.4	Influence of Participant Background	100
7.2.5	Qualitative Feedback	101
7.3	Track B: Workflow Comparison	102
7.4	Track C: Algorithmic Results	103
7.4.1	Benchmark Setup	104
7.4.2	Scalability	104
7.4.3	Time Granularity	107
7.4.4	Warm-Start	109
7.4.5	Concurrency Limits	110
7.4.6	Result Stability	111
7.5	Case Study: PS YETS 2025–26	112
7.5.1	Template Coverage	112
7.5.2	Schedule Reproduction	112
7.5.3	Task-Level Agreement	113
7.6	Limitations	114
7.7	Summary	115
8	Conclusions	116
8.1	Context and Synthesis	116
8.2	Research Questions and Objectives	117
8.3	Future Work	118
	References	121
A	Evaluation Instrument	128
A.1	Introduction	128
A.2	Informed Consent Form	128
A.2.1	Introduction	128
A.2.2	Voluntary Participation	128
A.2.3	Experiment Overview	129
A.2.4	Data Collection	129

A.2.5	Data Usage and Confidentiality	129
A.2.6	Contact Information	129
A.2.7	Consent	129
A.3	Pre-test Questionnaire	130
A.4	Task Instructions and Expected Outcomes	130
A.5	Task Ratings Grid	133
A.6	Post-test Questionnaire	133
A.6.1	System Usability Scale (SUS)	134
A.6.2	Open-ended questions	134
A.7	Track B Workflow Comparison Questionnaire	134
B	Collected Data	136
B.1	Participant Demographics	136
B.2	Planning Tools Currently Used	136
B.3	Task Performance	136
B.4	System Usability Scale Responses	139
B.5	Open-ended Feedback Themes (Track A)	139
B.6	Track B: Workflow Comparison Responses	139
B.7	Algorithmic Benchmark Data	139

List of Figures

2.1	“LS2 period” timeline	11
2.2	Scheduling Tools architecture	12
2.3	Gantt chart example in Scheduling Tools	12
2.4	The anatomy of a Gantt chart: each task is a horizontal bar on a shared time axis, whose position and length encode its start and duration, and links between bars represent dependencies.	15
2.5	Illustration of the visual information-seeking mantra of (Shneiderman, 1996) applied to a plan: overview first, zoom and filter, then details on demand.	16
3.1	Identification and filtering pipeline for the literature review.	21
3.2	Publication year distribution of selected records.	22
3.3	Example of a Gantt chart	23
3.4	LiveGantt interface: aggregation, interaction history, and details	24
3.5	Spring chart vs. Gantt chart for two schedule states	25
3.6	Project network with explicit resource nodes (hierarchical layout)	27
3.7	Task dependency types: FS, SS, FF, SF	28
3.8	CasCADE 4D concept: mapping time to a spatial axis	29
3.9	Chronographic schedule model for site congestion monitoring	30
3.10	PlanningVis interface with coordinated views	32
4.1	The two application flows: template capture (A) and schedule composition (B). Rounded blue boxes are actions the coordinator performs, rounded orange boxes are automated steps, and gray rectangles are artifacts.	51
4.2	The template as a reusable, parameterized blueprint. Authored once, a template carries a hierarchy of tasks and a set of named parameters; it is then instantiated any number of times, each instance binding the parameters — here a location and a team — to concrete values and placed on the calendar. The same template is reused both within a single stop, once per magnet, sector, or system, and across successive stops.	55
4.3	High-level architecture of the prototype.	57
4.4	The opaque solver response on an infeasible model: a bare INFEASIBLE status with no indication of the cause.	62
5.1	Capturing a reusable template from an existing plan.	67
5.2	Instantiating templates into a schedule, singly and in batch.	69
5.3	Defining placeholders in a template and binding them at instantiation.	70
5.4	Authoring a placeholder: declaring it on the template and marking, task by task, the fields it stands for.	71

5.5	Bridging the working-minute and calendar-minute domains with two compact lookup tables.	73
5.6	The conflict resolver: structured violations, each with a suggested one-click fix. .	76
5.7	The schedule page as a whole: the enriched timeline at the center, with the composition, optimization, and export controls in the side panel.	77
5.8	The visual notation for scheduling constraints and dependencies, with its legend. .	77
5.9	The concurrency heatmap beneath the timeline: a violated limit (red), a limit exactly reached (orange), and a respected limit, with the per-day detail shown on demand.	78
5.10	The enriched timeline: the before/after ghost overlay and the reason-annotated change list.	79
5.11	Reviewing an optimization: partial acceptance, manual override, and live violation feedback.	80
5.12	An exported schedule opened in Microsoft Project.	81
7.1	Age distribution (left) and self-reported Microsoft Project experience (right) of the twenty participants.	96
7.2	Distribution of time on task across the twenty participants. Boxes show the median and quartiles, whiskers the range, the diamond the mean, and the dots the individual participants.	97
7.3	Planned budget against actual mean time on task, for tasks T1 to T8.	97
7.4	Total errors and moderator assists per task, across all twenty participants. . . .	98
7.5	Total errors and moderator assists per participant, summed over the eight tasks. .	98
7.6	Distribution of perceived difficulty (top) and usefulness (bottom) for each task, on five-point scales. Boxes show the median and quartiles, whiskers the range, the diamond the mean, and the dots the individual participants.	99
7.7	System Usability Scale score for each participant. The dashed line marks the sample mean of 81.5; the dotted line marks the accepted industry mean of 68. . .	100
7.8	Distribution of SUS scores by participant group, for the three background contrasts. Boxes show the median and quartiles, whiskers the range, and the diamond the mean; the annotation gives the Mann–Whitney p -value.	101
7.9	Total time on task (top) and total errors (bottom) by participant group, for the three background contrasts. Box elements and annotation as in Fig. 7.8.	102
7.10	Scalability of solve time (top) and makespan (bottom) with problem size, for the denser Template 1 (left) and the sparser Template 2 (right). Each configuration is solved three times; markers distinguish optimal, feasible, and timed-out runs. . .	106
7.11	Mean makespan over three runs of four jointly scheduled instances of the sparser template as the number of inter-instance dependencies grows. Even a few links push the plan well above the parallel value, and a few tens saturate it near the serial regime, below the full chain; the bar at eight links spans the three runs. . .	107
7.12	Solve time at three time granularities. Finer slots cost solver time with no gain in makespan, which is identical across the three settings wherever optimality was proven.	108
7.13	Distribution of the 97 leaf-task durations in the benchmark template, in working days. The shortest task is half a working day and every value is a multiple of that unit.	108

7.14	Wall-clock time of the warm-start modes, decomposed into the solver’s search time and the hint-construction overhead, for the two templates. The day-level hint’s overhead outweighs the search time it saves; disabling the warm-start is the safe default.	109
7.15	Makespan (left) and solve time (right) as a function of the concurrency limit, for a location and a team constraint set, against the unconstrained baseline. Tighter limits lengthen the schedule and increase solve time; both relax to the baseline once the limit reaches three.	110
7.16	Makespan of two jointly scheduled instances of the sparser template under a team limit, a location limit, and both at once, against the unconstrained baseline. The two limits compound: constraining both yields a longer plan than either alone, until both reach a limit of three.	111
7.17	The PS YETS 2025–26 case study before and after optimization. At the template-group level the optimizer reproduces the overall envelope (a, b); at the task level it reorders tasks within that envelope (c).	113

List of Tables

2.1	Example schedule	8
2.2	Constraint categories	9
2.3	Types of programmed stops	10
2.4	Planning entities and their common attributes	14
2.5	Planner questions and implied information needs	15
3.1	Number of Papers Retrieved per Database and per Query	19
3.2	Screening Criteria for Study Selection	20
3.3	Number of reviewed publications by type and research topic.	20
3.4	Comparison of visual analytics patterns for interactive planning.	34
3.5	Comparison of constraint specification interface strategies.	36
3.6	Summary of visualization approaches.	37
3.7	Comparison of MILP and CP.	39
3.8	Population-based families in scheduling.	43
3.9	Decision matrix for selecting optimization approaches in constrained scheduling (synthesized from the families reviewed in Sections 3.3.1–3.3.3).	48
3.10	Existing systems against the four integrated capabilities	49
4.1	Functional and non-functional requirements	54
4.2	Scheduling constraint types supported by the optimizer and their encoding. . . .	60
5.1	Principal technologies of the prototype.	65
5.2	Violation typologies identified by the infeasibility diagnostics, their detection, and the suggested one-click fix.	76
6.1	Independent variables of the Track C benchmark and the values tested.	91
6.2	Mapping of in-test tasks and algorithmic experiments to the research questions. .	92
B.1	Demographic and background data for the twenty Track A participants. Group: Coord. denotes a planning coordinator and Non a non-planning member. CERN and MSP are years at CERN and years of Microsoft Project experience; Plan. is self-rated familiarity with the CERN planning workflow (1–5) and AD/EL. is self-rated familiarity with the AD/ELENA facility (1–5).	137
B.2	Tools and techniques currently used by each participant for planning.	137
B.3	Time on task in minutes for tasks T1–T8, with the per-participant total.	138
B.4	Per-task ratings for each participant, shown as difficulty / usefulness, each on a five-point scale (difficulty: 1 = very easy, 5 = very hard; usefulness: 1 = not useful, 5 = very useful).	138
B.5	Per-item System Usability Scale responses (1–5) for each participant.	139

B.6	Thematic summary of the Track A open-ended feedback, with the participants who raised each theme.	140
B.7	Track B responses to the three scaled questions (1–5). Confidence: that the prototype could produce a schedule at least as good as the manual one; Speed: expected speed relative to the current workflow; Benefit with complexity: expected change as schedules grow more complex.	141
B.8	Thematic summary of the Track B open-ended responses (seven coordinators). . .	141
B.9	Scalability versus problem size, by inter-instance dependency structure.	142
B.10	Time granularity. The optimal makespan is identical across granularities.	143
B.11	Warm-start, by Phase 1 mode.	143
B.12	Concurrency limits, for $N = 2$ independent template instances (194 leaf tasks, no inter-instance chain), by constraint set and limit.	144
B.13	Makespan under growing inter-instance coupling, for $N = 4$ independent sparse-template instances (388 leaf tasks). Each of the K links is a finish-to-start dependency between randomly chosen tasks of two different instances; each count is solved three times, with a fresh random choice of links.	144
B.14	Makespan under team and location concurrency limits applied separately and together, for $N = 2$ independent sparse-template instances (194 leaf tasks). Each configuration is solved three times.	144

List of Acronyms

2D	Two-dimensional
3D	Three-dimensional
4D	Four-dimensional
ACO	Ant Colony Optimization
AD	Antiproton Decelerator
ALAP	As Late As Possible
API	Application Programming Interface
ASAP	As Soon As Possible
BIM	Building Information Modeling
CAD	Computer-Aided Design
CERN	European Organization for Nuclear Research (<i>Conseil Européen pour la Recherche Nucléaire</i> , original name)
CP	Constraint Programming
CP-SAT	Constraint Programming over Boolean Satisfiability (Google OR-Tools solver)
CPM	Critical Path Method
CPN	Colored Petri Nets
CPU	Central Processing Unit
CSV	Comma-Separated Values
DE	Differential Evolution
ELENA	Extra Low ENergy Antiproton ring
EYETS	Extended Year-End Technical Stop
FF	Finish-to-Finish
FNET	Finish No Earlier Than
FNLT	Finish No Later Than
FS	Finish-to-Start
GA	Genetic Algorithm
GB	Gigabyte
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IHTC	Integrated Healthcare Timetabling Competition
ILP	Integer Linear Programming
IMPACT	Intervention Management Planning and Coordination Tool
JSON	JavaScript Object Notation
KPI	Key Performance Indicator

LHC	Large Hadron Collider
LNS	Large Neighborhood Search
LOB	Line of Balance
LS	Long Shutdown
MFO	Must Finish On
MFS	Minimal Forbidden Sets
MILP	Mixed Integer Linear Programming
MRCPSP	Multi-mode Resource-Constrained Project Scheduling Problem
MRO	Maintenance, Repair, and Overhaul
MS	Microsoft
MSO	Must Start On
NP	Non-deterministic Polynomial time
NSGA-II	Non-dominated Sorting Genetic Algorithm II
OVC	Ordered Visiting Constraints
PDF	Portable Document Format
PDM	Precedence Diagramming Method
PERT	Program Evaluation and Review Technique
PPO	Proximal Policy Optimization
PS	Proton Synchrotron
PSO	Particle Swarm Optimization
PSPLIB	Project Scheduling Problem Library
RCMPSP	Resource-Constrained Multi-Project Scheduling Problem
RCPSP	Resource-Constrained Project Scheduling Problem
REST	Representational State Transfer
RL	Reinforcement Learning
RQ	Research Question
SF	Start-to-Finish
SGS	Schedule Generation Scheme
SMT	Satisfiability Modulo Theories
SNET	Start No Earlier Than
SNLT	Start No Later Than
SPS	Super Proton Synchrotron
SS	Start-to-Start
SUS	System Usability Scale
TS	Technical Stop
XML	Extensible Markup Language
YETS	Year-End Technical Stop

Chapter 1

Introduction

Scheduling is a decision-making process, used regularly in numerous manufacturing and service industries (Leung et al., 2004). It is responsible for the allocation of resources to activities over defined time periods with the goal of optimizing one or more objectives. Efficient scheduling is vital not only to ensure the timely completion of tasks but also to reduce costs, mitigate risks, and maximize the availability of sophisticated systems for industrial use.

As modern systems evolve, they become more sophisticated and interconnected, and traditional scheduling often struggles to handle the growing number of variables and constraints (Hartmann & Briskorn, 2010). This situation calls for advanced methods, such as optimization algorithms, with the aim of improving decision-making (Hartmann, 2021). At the same time, to interpret complex scheduling data and understand dependencies, it is necessary to employ a simple and intuitive visual language capable of illustrating such information effectively (Munzner, 2014).

This dissertation aims to study these challenges by investigating how a platform that combines visual language with optimization techniques can improve the scheduling of programmed stops at the European Organization for Nuclear Research (CERN). The following sections outline the motivation behind this study, along with the key questions it seeks to address.

1.1 Context

CERN¹ is an international organization located on the Franco-Swiss border near Geneva. CERN's origins can be traced back to the 1940s, when a small number of visionary scientists in Europe and North America identified the need for Europe to have a world-class physics research facility. There, physicists and engineers use the world's largest and most complex scientific instruments to study the basic constituents of matter by accelerating particles and colliding them at near-light

¹CERN official website at <https://home.cern>

speeds. This is only possible with the use of specialized facilities, such as the Large Hadron Collider (LHC)² and the Proton Synchrotron (PS)³, together with the other machines of the accelerator complex, including the Linac4⁴, the PS Booster⁵, and the Super Proton Synchrotron (SPS)⁶.

At CERN, various experiments are performed on a daily basis, but for the machines to work correctly, they need regular maintenance and check-ups. Usually, these maintenance tasks, called “Technical Stops” (TS), take hours or days to complete and use different subsections of a machine, where more than one group can be involved simultaneously. Occasionally, these machines need to undergo “Year-End Technical Stops” (YETS), lasting several months (Vergara Fernandez et al., 2022). Additionally, after long periods of operation, they also need to undergo repairs and upgrades that can last for years, known as “Long Shutdowns” (LS).

The detailed planning of these operations is a demanding coordination exercise: multiple teams have to collaborate, and all constraints must be satisfied in order to ensure smooth execution. In this context, a platform that integrates visual representations of tasks with optimization capabilities can reduce errors and increase the efficiency of these maintenance periods.

1.2 Motivation

Currently, CERN provides the “Scheduling Tools” web platform to support the visualization and analysis of maintenance activities (Matos, 2025). Although this tool offers multiple ways to visualize maintenance tasks, such as the Gantt View, Calendar View, Two-dimensional (2D) Top View, and a Three-dimensional (3D) Visualization system, detailed scheduling of programmed stops is still a predominantly manual procedure. At present, CERN relies on Microsoft (MS) Project for this process, which, although having various functionalities, lacks some required visual feedback and interactive support for reasoning about scheduling constraints.

Planning a Technical Stop involves coordinating multiple interdependent tasks and constraints related to time, space, resources, and safety. Manual planning often requires schedule adjustments as new limitations or tasks emerge, which may become complex and error-prone at this level. This increases the risk of conflicts and inefficiencies and may even cause some avoidable delays in the overall process. The manual effort is further compounded by repetition: specific interventions, such as YETS, typically decompose into a standard baseline of ordered tasks with consistent durations and responsible teams. Yet, these recurring task groups are often reconstructed by hand for every new plan.

These limitations motivate research on improving the planning of programmed stops through more expressive, interactive visual representations of scheduling constraints and their conflicts, and computational methods that help planners explore feasible, high-quality schedules under realistic constraints, complementing rather than replacing existing practices.

²LHC overview at <https://home.cern/science/accelerators/large-hadron-collider>

³Proton Synchrotron overview at <https://home.cern/science/accelerators/proton-synchrotron>

⁴Linac4 overview at <https://home.cern/science/accelerators/linear-accelerator-4>

⁵PS Booster overview at <https://home.cern/science/accelerators/proton-synchrotron-booster>

⁶SPS overview at <https://home.cern/science/accelerators/super-proton-synchrotron>

Given the limited availability of engineering and research time at CERN, improvements in planning support can have a direct impact on productivity. Moreover, inadequate planning may propagate delays and force costly rework, while efficient planning leads to increased productivity.

1.3 Problem

Although existing tools are beneficial for tracking maintenance tasks, they offer limited support for reasoning about complex scheduling problems under interacting constraints (Desai et al., 2023). Particularly, widely used tools such as MS Project focus primarily on task sequencing and resource allocation but provide limited formal methods for modeling and analyzing interactions among temporal, spatial, resource, safety, and operational constraints within a unified framework.

In large-scale maintenance planning contexts such as CERN, planning decisions involve a dense network of interconnected dependencies. As a result, a single local modification to a plan may propagate across the schedule, affecting feasibility in ways that are difficult to predict using traditional representations. In addition, external factors, including resource availability, unforeseen failures, or evolving operational conditions, complicate the reasoning process (Vergara Fernandez et al., 2022). Existing tools offer limited support for understanding how these uncertainties impact global schedule feasibility.

From a methodological perspective, this reveals a gap between scheduling representations and the complexity of constraint-driven planning problems. While visual tools prioritize readability and manual inspection, optimization techniques operate on abstract models that are not directly accessible to planners. This separation constrains the planner's ability to reason about constraint interactions, interpret optimization outcomes, and inspect the consequences of modifications in an informed manner.

Therefore, this work addresses this gap between visual and optimization-based scheduling support, focusing on the temporal and resource dimensions. It examines how combining visualization and optimization can enhance the understanding, analysis, and control of scheduling problems in realistic settings.

1.4 Objectives and Contributions

The main objectives of this dissertation are to investigate which combinations of visualization and optimization are most appropriate for supporting scheduling in large-scale maintenance environments such as CERN. This work aims to establish and evaluate approaches that support planners in representing constraints expressively and in generating and revising schedules under realistic conditions of scale and constraint interaction.

Therefore, these objectives raise the following research questions (RQs):

- **RQ1:** What visual representation techniques and ‘visual language’ principles can be employed to model, display, and interact with complex scheduling constraints in task and project planning systems?

- **RQ2:** How can constraint-rich project scheduling in this context be modeled and solved, and what trade-offs between schedule quality and computational cost does the resulting approach exhibit?

Alongside the research questions, the work sets four concrete objectives that the evaluation tests directly: to validate the template abstraction (O1), to characterize the optimizer (O2), to establish the prototype’s usability (O3), and to assess its benefit against the current planning workflow (O4). These are defined in full with the evaluation protocol in Chapter 6 and revisited in the conclusions.

Underlying both questions is the observation that the recurring structure of these interventions can be captured once and reused. The approach taken in this work therefore centers on a reusable *template* abstraction that serves as the shared substrate for the two research questions: it provides the structured, interactive representation through which scheduling constraints are expressed (RQ1) and the input on which the optimization operates (RQ2).

To answer these questions, this dissertation analyzes the literature to identify relevant visualization and optimization techniques, formalizes the CERN planning problem with its constraints and objectives, and develops a prototype that integrates the selected methods. The prototype is then evaluated for its ability to reason about feasibility and conflicts and to produce feasible schedules under realistic requirements.

In these terms, this dissertation contributes to constraint-based scheduling and to the visual analytics of planning through an integrated, template-centric approach to recurring maintenance interventions, resting on two main contributions. The first, and the conceptual core, is the reusable, parameterized template abstraction together with the workflow built on it, which closes the loop from authoring a plan once, through optimization, to a reviewed export — a loop that conventional planning tools, supporting inspection or optimization but rarely both over a reusable abstraction, leave open. The second is a human-in-the-loop review interface, paired with an explainable infeasibility diagnosis that turns an opaque solver rejection into typed, fixable violations; it is through this interface that the coordinator supplies the experiential knowledge that lies outside any formal model. The optimizer, by contrast, is deliberately conventional: it reconciles working and continuously-elapsing time on a common scale using only standard constraint forms, rather than a bespoke calendar procedure. This approach is realized and evaluated for the programmed stops at CERN.

1.5 Document Structure

Beyond this introduction, this dissertation is organized into seven additional chapters. Chapter 2 provides the necessary background, introducing the CERN shutdown planning context and the main foundations in scheduling and visualization that support the remainder of the work. Chapter 3 presents the literature review, describing the adopted review methodology and synthesizing the state of the art, covering both optimization methods for constrained scheduling and visualization approaches for planning and decision support. Chapter 4 specifies the solution: the flows it

supports, its architecture, the template abstraction, the scheduling problem, and the design of the visualization and of the review workflow, together with the main design decisions behind them. Chapter 5 describes how that specification is realized, covering the technology stack, the template engine, the encoding of the scheduling problem in the solver, and the interactive interface. Chapter 6 presents the evaluation protocol, defining the objectives, tracks, and tasks used to assess the prototype. Chapter 7 reports the results of that evaluation. Finally, Chapter 8 draws conclusions and outlines directions for future work.

Chapter 2

Background

This chapter provides the domain and conceptual background needed for the rest of this work. Section 2.1 introduces the scheduling vocabulary (constraints and objectives) required to formalize the problem. Section 2.2 describes the CERN planning context and the main entities involved in programmed stops. Section 2.3 summarizes visualization and interaction concepts that motivate the information needs and interface requirements considered later.

2.1 Scheduling Foundations

This section introduces the scheduling foundations used in this dissertation. It first defines the basic scheduling concepts and representations, and then discusses how constraints and objectives characterize feasibility and schedule quality.

2.1.1 Scheduling Contexts and Problem Types

Scheduling arises in many application contexts, each typically linked to a small set of canonical problem types. Tournaments are fixtured in sports competitions (Kendall et al., 2010), course and examination timetables are constructed in education (Schaerf, 1999), and workforce rostering assigns staff to shifts under labor rules and coverage requirements, with preferences modeled as soft constraints (Burke et al., 2004; Ernst et al., 2004). In production, jobs are assigned to machines within capacity limits, leading to classical environments such as single-machine and parallel-machine scheduling, as well as shop settings such as flow-shop and job-shop scheduling (Brucker, 2007).

A standard way to describe deterministic scheduling problems is the three-field notation $\alpha|\beta|\gamma$, where α specifies the processing environment (e.g., one machine, parallel machines, shop setting), β lists relevant constraints (e.g., release dates, precedence constraints, calendars), and γ defines the objective (e.g., makespan $C_{\max}$, total completion time $\sum C_i$) (Brucker, 2007; Graham et al., 1979). For instance, $1|prec|C_{\max}$ denotes a single-machine problem with precedence relations and the objective of minimizing the makespan (Graham et al., 1979).

This dissertation addresses maintenance planning by scheduling interdependent tasks over a finite time horizon. A feasible plan must satisfy precedence relations, resource limits, calendar constraints, and other requirements. These considerations motivate the core concepts and representations introduced next.

2.1.2 Core Concepts and Representations

Scheduling is the process of assigning time decisions to a set of tasks while satisfying all the requirements (Pinedo, 2022). In maintenance planning, a schedule is typically defined over a set of activities that must be executed within a given time horizon, subject to precedence relations, limited resources, and other constraints that influence the overall objective (Demeulemeester & Herroelen, 2002).

Let $V = \{1, \dots, n\}$ represent the set of activities. Each activity $i \in V$ has a duration $p_i > 0$ and is assumed to be non-preemptive unless stated otherwise (Pinedo, 2022). A schedule assigns a start time S_i to every activity i , with completion time $C_i = S_i + p_i$. A schedule is feasible when it satisfies all constraints between activities.

In order to describe schedules, two complementary representations are often used. First, a timeline representation places activities on the same time axis to show both their start and end times. Second, a precedence representation shows dependencies between tasks as a directed graph $G = (V, E)$, where an arc $(i, j) \in E$ indicates that activity i must precede activity j under a given relation (Demeulemeester & Herroelen, 2002). These representations are complementary, as timelines show temporal overlap, while graphs emphasize the ordering and structure of the dependencies.

Regarding time, some temporal terms are defined below for better understanding of the domain (Demeulemeester & Herroelen, 2002; Pinedo, 2022):

- **Duration vs. effort:** duration is elapsed time; effort is labor content (e.g., person-hours) that may be distributed depending on staffing.
- **Milestone:** an event that represents a required time point (e.g., finish of an overall plan).
- **Baseline vs. current schedule:** a baseline is a reference schedule, while the current schedule is the updated plan after incorporating revisions (Project Management Institute, 2011).

Calendars are also part of scheduling. In practice, not all time periods are usable, as there are non-working intervals and limited access windows. Time windows are used to express earliest/latest execution limits and gather availability restrictions (Dorndorf, 2012).

Below is a simple example to clarify the terminology:

Table 2.1: Example schedule

ID	Task/Activity	Duration	Resource	Notes
A	Access preparation	2 h	Team T1	Must be completed before work starts in that area
B	Component replacement	6 h	Team T1	Requires same team as activity A
C	Functional test	3 h	Team T1	Can only start after B finishes
D	Documentation update	1 h	Team T2	Must occur after C
E	Safety approval	2 h	Safety Officer	Must be completed before A

A precedence graph for this example could include edges $G = \{(E,A), (A,B), (B,C), (C,D)\}$, as those dependencies can be inferred from Table 2.1. An example of a calendar constraint could be that the interval 13:00 to 14:00 and the weekends are unavailable for work, and an example of a resource constraint is that Team T1 can only do one task at a time, meaning that activities A, B, and C could not be done in parallel. The purpose of this example is to clarify that feasibility depends not only on precedence dependencies but also on other constraints.

2.1.3 Constraints and Objectives

A constraint is a rule that restricts the set of feasible schedules. So, a schedule must satisfy all the conditions to be feasible. However, schedules commonly have two types of constraints: hard and soft (Brucker et al., 1999; Hartmann & Briskorn, 2010; Rossi et al., 2008).

Below is the difference between the two types of constraints:

- **Hard constraints:** must be satisfied, and violating one constraint of this type generates an infeasible schedule.
- **Soft constraints:** do not need to be satisfied; however, they are typically desirable and are used to express preferences; violating a soft constraint is permitted but often penalized or otherwise reflected in lower schedule quality.

According to the above explanation, feasibility is defined by hard constraints, while quality is evaluated via objectives and often by penalties for violating soft constraints.

Often, constraints can be divided into categories as seen in Table 2.2:

Table 2.2: Constraint categories (adapted from (Demeulemeester & Herroelen, 2002; Dorndorf, 2012; Pinedo, 2022)) with examples inspired by the CERN planning context (based on (Vergara Fernandez et al., 2022)).

Category	Examples
Temporal	Precedence relations (e.g., finish-to-start); minimum/maximum lags; time windows; milestones; access window constraints
Resources	Team capacity; qualification requirements; equipment availability; shared tools; resource calendars
Spatial	Area exclusivity; occupancy limits; interference between nearby activities; restricted zones
Safety or Operational	Permit readiness; mandatory approvals; training prerequisites; procedural sequencing

With constraints defined, the next concept is the objective. Objectives quantify what makes one schedule better than another. Often, schedules do not have a single objective, but several to fulfill (Brucker et al., 1999; Hartmann & Briskorn, 2010). Thus, this section provides examples of common objectives:

- **Makespan / total duration:** minimize overall completion time of the schedule.
- **Delays / lateness:** minimize lateness relative to due dates or milestones previously defined; a due date is a target whose violation is penalized, whereas a deadline is a hard limit that a feasible schedule may not exceed (Pinedo, 2022).
- **Resource usage:** reduce peaks, smooth workload, or minimize overloads or exposure.
- **Conflict reduction:** minimize the number and/or severity of constraint violations.

In conclusion, this section clarified schedules, constraints, objectives, and the simple vocabulary that will be used in the following chapters. The next section instantiates these concepts in the specific CERN context, highlighting the entities, constraints, and workflows that motivate the entire planning problem.

2.2 CERN Planning Context

CERN programmed stops focus maintenance and upgrade work into planned downtime windows, which creates a problem that involves several dimensions under strict constraints (Vergara Fernandez et al., 2022).

Planning becomes more constrained as interventions are distributed across locations with access constraints that require specialized equipment and teams. Usually, planning is iterative because plans evolve as new information enters the scheme, priorities change, and this creates a dynamic environment around the whole planning process (Vergara Fernandez et al., 2022).

2.2.1 Programmed Stops

A Programmed Stop is a dedicated time window in which the conditions of CERN machines are intentionally set to allow for maintenance or upgrades, with interventions varying depending on the objective and the stop (Vergara Fernandez et al., 2022). These are essential, as they define the temporal limits within which various tasks or interventions must be prepared, executed, and coordinated while still respecting all constraints. A defining characteristic of these interventions is their recurrence: across successive stops and across similar pieces of equipment, a given intervention decomposes into broadly the same ordered group of tasks, performed by the same teams and with comparable durations, even though each plan is currently assembled from scratch.

There are four types of programmed stops, which differ in duration and scope, and are summarized in Table 2.3:

Table 2.3: Types of programmed stops (adapted from (Vergara Fernandez et al., 2022)).

Type	Order of magnitude	Planning level	Typical activities
Technical Stop (TS)	hours or days	local / operational	localized maintenance, short access windows
Year-End Technical Stop (YETS)	weeks or months	mixed	coordinated maintenance across systems
Extended Year-End Technical Stop (EYETS)	months	mixed	deeper interventions, broader coordination across systems
Long Shutdown (LS)	years	portfolio	major upgrades across all machines

This classification is relevant because it reflects the number of interventions, the diversity of teams and/or systems, and the coordination effort required during execution (Vergara Fernandez et al., 2022).

Technical stops (TS) are characterized by tightly constrained execution windows, typically spanning several hours to a few days (Vergara Fernandez et al., 2022). In contrast, longer interventions such as the Year-End Technical Stop (YETS) and the Extended Year-End Technical Stop (EYETS) extend over several weeks or months and support a wider array of activities, including system upgrades¹. These extended periods increase the available execution window but also require broader coordination to manage multiple concurrent activities within established access and safety protocols (Vergara Fernandez et al., 2022). Additionally, long shutdowns (LS) facilitate the implementation of major modifications and the execution of large-scale project portfolios, all while upholding rigorous quality and safety standards. Figure 2.1 provides a visual timeline of these maintenance periods, specifically highlighting the LS2 period at CERN (Vergara Fernandez et al., 2022).

This planning problem can be characterized along several dimensions (Demeulemeester & Herroelen, 2002; Pinedo, 2022; Vergara Fernandez et al., 2022):

¹ More information at <https://cerncourier.com/a/machine-technical-stop-in-full-swing/>

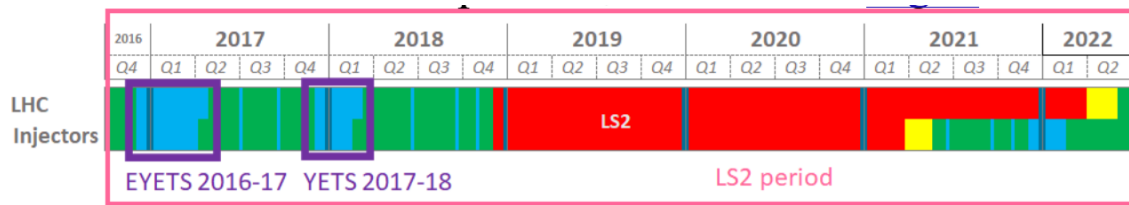


Figure 2.1: “LS2 period” timeline. Taken from (Vergara Fernandez et al., 2022).

- **Time:** stop duration, milestones, time windows, working periods, and other time-related constraints.
- **Tasks and dependencies:** precedence relations, access ordering, lags, coupling between subsystems, and related dependency constraints.
- **Resources:** teams, skills, equipment, capacity limits, availability calendars, and additional resource constraints.
- **Space and access:** work zones, restricted areas, occupancy limits, interference between nearby interventions, and further access-related constraints.
- **Safety:** access authorization, training prerequisites, permits to work, and other safety requirements.
- **Change and re-planning:** evolving scope, updates to plans during preparation and execution, and additional consequences for feasibility and resource assignment.

In summary, CERN planning spans multiple levels of detail, depending on the type of stop. However, all conditions must be met for planning to remain feasible.

2.2.2 Planning Workflow and Data

At CERN, activity schedules are created using external planning tools such as MS Project and subsequently integrated into a centralized web platform to enable cross-facility inspection. The Scheduling Tools system, as shown in Fig. 2.2, consists of a relational database for storing projects and tasks, backend services that provide access to planning data, and a frontend application offering multiple views for monitoring and coordination (Vergara Fernandez, Rousis, et al., 2023).

In the current setup, schedule data is synchronized daily (unidirectionally) from the planning tool environment into the Scheduling Tools database, after which the web application provides access to planned activities across facilities (Vergara Fernandez, Rousis, et al., 2023). In this arrangement the web application is read-only: it is a window onto plans authored elsewhere, so coordinators inspect and monitor schedules through it, while the authoring itself, and any rescheduling, happen externally in Microsoft Project. This inspection-only boundary is the one the present work sets out to move, by bringing authoring, optimization, and review inside the platform. Figure 2.3 presents an example of this functionality using the Gantt interface.

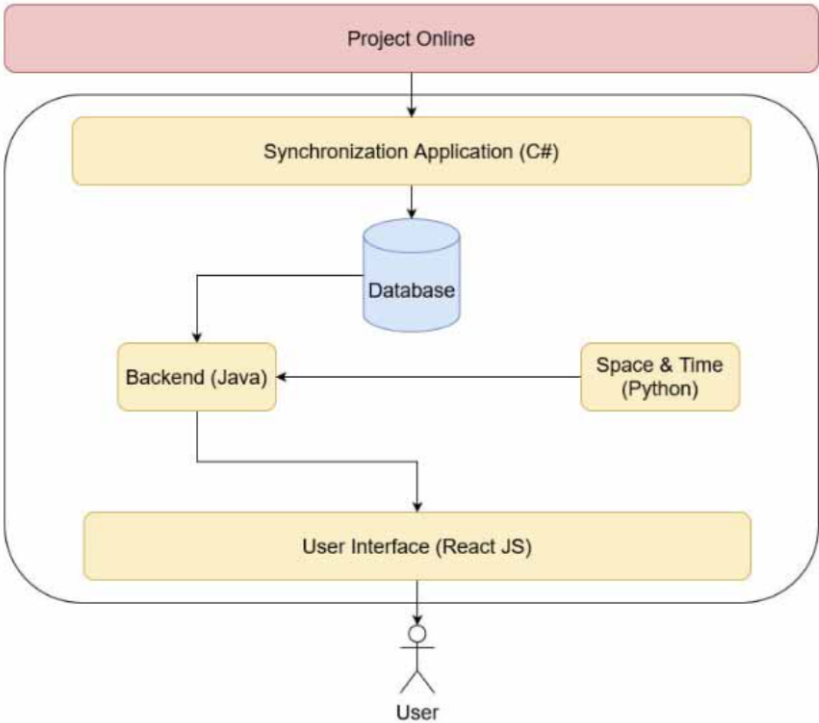


Figure 2.2: Scheduling Tools architecture and data flow (adapted from (Vergara Fernandez, Rousis, et al., 2023)).

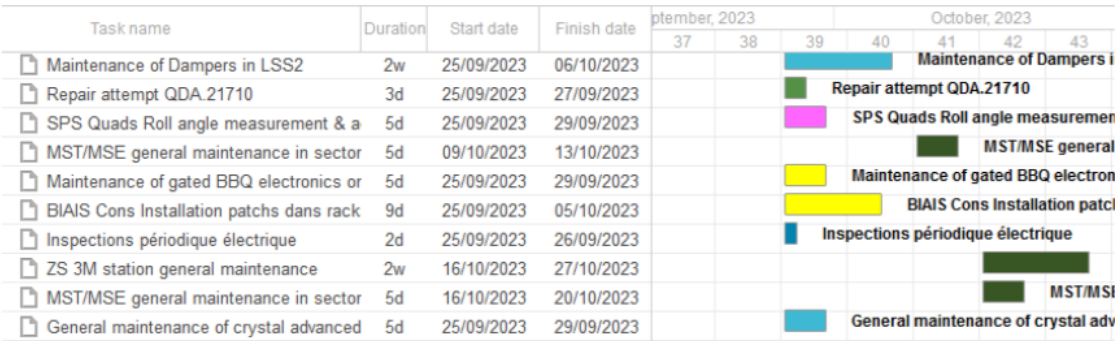


Figure 2.3: Example of a Gantt Chart in Scheduling Tools. Taken from (Vergara Fernandez, Rousis, et al., 2023).

Planning responsibilities are distributed across organizational roles. Schedule updates are typically provided by the relevant activity owners and planning contributors, while CERN technical coordination functions oversee cross-system coordination throughout programmed stops (Perrot et al., 2022). The Scheduling Tools project team supports the shared platform and consolidates user needs across facilities (Vergara Fernandez, Rousis, et al., 2023). In the scheduling literature this planning role is termed the *planner*; in the CERN context it is carried out by the *coordinators* responsible for each programmed stop, the term this dissertation adopts from Chapter 4 onward.

In addition to the scheduling data flow, planning at CERN is constrained by formal declaration, approval, and access procedures that must be satisfied before work can start (Vergara Fernandez et al., 2022). For example, work declaration and coordination are supported by formal processes and dedicated tools such as IMPACT, the Intervention Management Planning and Coordination Tool². Also, some restricted zones require training and authorization to ensure safety during operations^{3,4}.

Planning information in this setting can be described using a small set of entities and relationships, in which two entries are specializations of a third: calendars and dependencies are particular kinds of constraint (Demeulemeester & Herroelen, 2002; Pinedo, 2022; Project Management Institute, 2011):

- **Task (Activity):** a unit of work that can be planned, monitored, and coordinated.
- **Resource:** a human or technical capability required to execute activities, possibly with limited capacity.
- **Location:** a physical work area whose access and occupancy conditions influence the schedule, often modeled as a resource or as an attribute of a task.
- **Constraint:** a rule that restricts which plans are feasible, with hard or soft semantics.
- **Calendar:** a constraint that restricts when tasks and resources are available, a specialization of the entry above.
- **Dependency:** a constraint that restricts the relative ordering of two tasks, likewise a specialization of constraint.
- **Objective:** a criterion used to compare plans and measure their quality.

For clarity, Table 2.4 summarizes the above entities and their representative attributes.

²The IMPACT tool at <https://impact.cern.ch> (access restricted to CERN users)

³More information available at <https://hse.cern/content/why-and-how-enrol-course>

⁴See more at <https://cem-mro.web.cern.ch/mandatory-courses-cern-site-and-radiation-areas-access>

Table 2.4: Planning entities and their common attributes (extracted from (Demeulemeester & Herroelen, 2002; Pinedo, 2022; Project Management Institute, 2011; Vergara Fernandez et al., 2022)).

Entity	Representative attributes
Task (Activity)	ID, name, duration, earliest/latest start/finish, status, hierarchy, priority
Dependency	type, lag, predecessor/successor
Resource	team/equipment, capacity, availability calendar, skills
Location	zone/installation, access permissions, occupancy limits, interference conditions
Calendar	working periods, non-working periods, access windows, shutdown rules
Constraint	category (temporal/resource/spatial/safety), parameters, hard vs. soft semantics
Objective	makespan/duration, lateness/delay, utilization peaks, conflict count, exposure

While this section has established the planning context and a small vocabulary from the domain, the next section discusses visualization in scheduling.

2.3 Visualization Foundations

This section introduces the foundations for visualization and interaction in scheduling. The goal of visualization is to clarify what information must be communicated from plans to the user, and which operations should be supported (Munzner, 2014).

Information visualization, in general, provides interactive visual representations of data designed to help people carry out analysis tasks more effectively: data is mapped to visual encodings such as position, length, and color, so that patterns, outliers, and relationships can be perceived directly rather than read item by item (Munzner, 2014). A visualization is therefore designed rather than given: the same data admits many representations, and an effective one is chosen by asking what data the users have, why they need to look at it, and how the encoding and the interaction serve that need (Munzner, 2014). Static images alone rarely suffice for data of any richness; interaction lets the user change what is shown, and at what level of detail, which is what makes large information spaces navigable (Shneiderman, 1996). The remainder of this section applies this lens to plans and schedules.

2.3.1 Plan Visualization Requirements

Maintenance is inherently multi-dimensional, encompassing factors such as time, resources, and space. The main objective is to present this information intuitively to planners, and also to the team members who execute the work and read the plan to know their own calendar. Therefore, aligning data, tasks, and visual idioms is fundamental to ensure that the representation directly supports the user’s informational needs (Munzner, 2014).

Thus, visualization should support:

- **Temporal structure:** timing, overlap, idle periods, milestones.

- **Dependencies:** precedences, propagation.
- **Resources:** allocation over time, capacity peaks.
- **Spatial context:** where work occurs, restricted zones, occupancy, interferences.
- **Conflicts and priorities:** violations and which tasks are the priority.
- **Scenario comparison:** baseline vs. current schedule; differences between alternatives.

Figure 2.4 grounds the first of these needs in the timeline representation used throughout this work, the Gantt chart: each task is a horizontal bar on a shared time axis, the position and length of the bar encode the task's start and duration, and dependencies are drawn as links between bars.

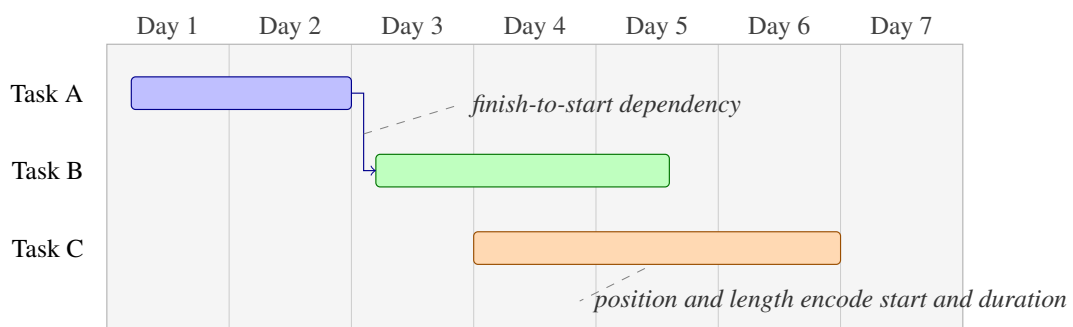


Figure 2.4: The anatomy of a Gantt chart: each task is a horizontal bar on a shared time axis, whose position and length encode its start and duration, and links between bars represent dependencies.

These needs imply that users must be able to alternate between overview and details, allowing the planner to be aware of all the information intuitively (Cockburn et al., 2009; Shneiderman, 1996).

Finally, planning is not only inspection but iterative decision-making. It is crucial for users to understand and interact with schedules in order to explore the best alternatives and reach decisions in complex environments (Keim et al., 2008).

Lastly, Table 2.5 presents an example of questions that visualization should allow users to answer and the information that they imply:

Table 2.5: Examples of planner questions and the information/interaction they imply (adapted from (Munzner, 2014; Shneiderman, 1996; Wang Baldonado et al., 2000)).

Planner question	Required information	Key interaction
What prevents activity A from starting earlier?	Precedences; time windows; resource availability	Trace dependencies; inspect constraints between tasks;
Where are conflicts expected in area B during a time window?	Spatial zones; access windows; overlapping tasks	Filter by time and area; highlight overlaps
Which team is overloaded during week C?	Resource assignments; capacity; calendar	Aggregate by team; sort

2.3.2 Interaction Concepts

Interaction is where planners explore, test alternatives, and make decisions. Below are common interaction techniques used in schedule visualization, required for plan analysis:

- **Filtering, searching, sorting:** progressive refinement of schedules, in order to view details that cannot be seen when overviewing the plan (Shneiderman, 1996).
- **Navigation and focus:** zooming/panning and focus+context to preserve global awareness while inspecting detail (Cockburn et al., 2009).
- **Linked views:** different but coordinated views across the distinct dimensions to reveal relationships (Wang Baldonado et al., 2000).
- **Details on demand:** inspection of activities, dependencies, constraints, and allocations without cluttering the overview schedule (Shneiderman, 1996).
- **What-if reasoning:** explore different scenarios, compare them, and know the changes that each one implies (Keim et al., 2008).

Figure 2.5 sketches how the first four of these techniques compose into the canonical visual information-seeking workflow: an overview of the whole plan first, then zooming and filtering into a region of interest, and details about an individual task only on demand (Shneiderman, 1996).

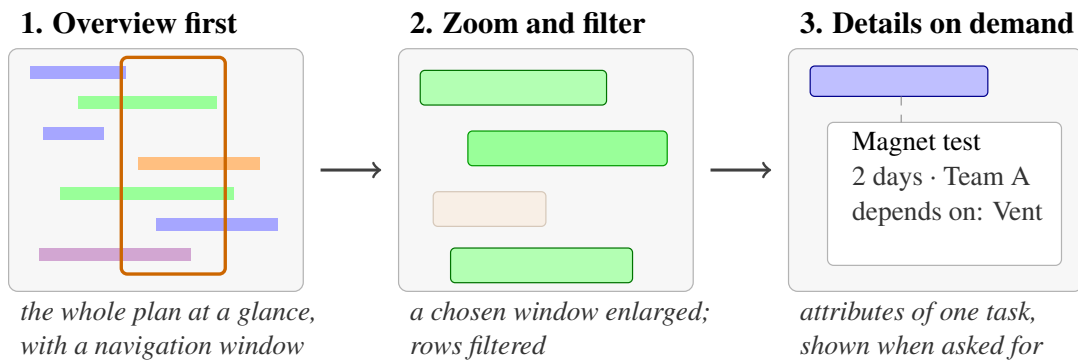


Figure 2.5: Illustration of the visual information-seeking mantra of (Shneiderman, 1996) applied to a plan: overview first, zoom and filter, then details on demand.

2.4 Summary

This chapter has established the domain context of CERN programmed stops, the scheduling vocabulary of tasks, constraints, and objectives, the categories of constraint that arise in practice, and the visualization and interaction concepts that an effective planning tool must support. The following chapter reviews the state of the art in scheduling optimization and in the visualization of plans and constraints, identifying the techniques and the gaps that motivate the approach taken in this dissertation.

Chapter 3

Related Work

To address the research questions defined in Section 1.4, the following chapter details the systematic literature review methodology in Section 3.1. Finally, Sections 3.2 and 3.3 examine the state of the art in visualization and optimization to identify critical gaps and opportunities for improving the scheduling of CERN's programmed stops.

3.1 Literature Review Process

Scheduling has two main subtopics. The first is optimization, the field that addresses mathematical techniques and algorithms used to generate the most effective and efficient schedule solutions based on defined constraints and objectives. The second is visualization, where the priority is the design of efficient user interfaces to view, interact with, and manipulate schedules for maximum control and user clarity. Moreover, the primary objective of this literature review is to identify and study the tools and techniques already utilized by others in both these areas and, crucially, to pinpoint critical research gaps and opportunities. To accomplish this, a hybrid literature review methodology was adopted. The initial part was exploratory, focused on identifying foundational articles and necessary keywords for both themes. The subsequent phase involved a systematic search across databases, choosing the most relevant publications based on predefined criteria.

3.1.1 Search Strategy

The following section outlines a structured approach to identify, filter, and select publications in various academic databases, ensuring the methodological consistency and replicability required of a systematic review. Two major digital databases were accessed to collect relevant conference papers, scientific articles, and books on the topic. These are IEEE Xplore¹, the digital library of the Institute of Electrical and Electronics Engineers (IEEE), and Scopus². These were selected as primary sources due to their relevance to engineering and computer science and their support for structured querying and export. The choice was deliberate: both index the engineering

¹The IEEE digital library, at <https://ieeexplore.ieee.org/>

²Elsevier's abstract and citation database, at <https://www.scopus.com/>

and computer science venues central to the topics under review, Scopus additionally covers the operations research literature through its Decision Sciences subject area, and both support the reproducible, filterable queries on which a documented search protocol depends. Restricting the systematic search to two databases is nonetheless a limitation; it is mitigated by the snowballing step described below, which was conducted mainly through Google Scholar³ and followed references regardless of where they are indexed. Later, references were systematically managed using Zotero⁴ and Rayyan⁵ to retain the most significant publications.

3.1.2 Identification Phase

The search started with two main queries, one on the visualization topic, and the other one on optimization:

- **QS1:** ((visual* OR "visual language" OR "graphical interface" OR diagram) NEAR/7 (scheduling OR "task planning" OR "project planning" OR "production planning")) AND (constraint OR precedence OR dependency OR resource OR allocation OR temporal OR spatial OR location OR equipment OR team OR safety OR hazard OR operational)
- **QS2:** ((planning OR scheduling) NEAR/5 (optimization OR "constraint programming")) AND (infrastructure OR industrial OR facility OR manufact* OR production OR maintenance OR "multi-project" OR construction) AND (((temporal OR spatial OR location OR equipment OR team OR safety OR hazard OR operational) NEAR/3 (constraint OR problem OR allocation OR task OR scheduling)) OR ("resource constrained" AND ("project scheduling" OR "task scheduling")))

These query strings encompass both main topics of the study. The wildcard "*" is used to match all terms referring to both visualization (e.g., visualizing, visual) and manufacturing (e.g., manufactories, manufacture). Furthermore, the "NEAR" operator is utilized to tighten the search, ensuring that the linked concepts appear in close proximity within the text. This technique is crucial to reduce false positives while maintaining high sensitivity.

The search terms had to be adapted to work with the two selected databases:

- **IEEE Xplore:** Default search query;
- **Scopus:** Uses "W" instead of "NEAR" operator, as they are equivalent. Search within "Article Title, Abstract, Author Keywords" and limited to the subjects of "Engineering", "Decision Sciences", "Mathematics" and "Computer Science".

³Google Scholar, at <https://scholar.google.com/>

⁴The Zotero reference manager, at <https://www.zotero.org/>

⁵The Rayyan systematic-review screening tool, at <https://www.rayyan.ai/>

Table 3.1: Number of Papers Retrieved per Database and per Query

	QS1	QS2	Total
IEEE Xplore	1108	1319	2427
Scopus	559	947	1506
Overall Total	1667	2266	3933

3.1.3 Screening Phase

The systematic search initially yielded a total of 3933 records, as detailed in Table 3.1. However, given the large number of records yielded in this phase, a cutoff strategy was applied. In each query and database, the titles of the articles were read, until there was no single article on a page that would be a possible candidate (25 articles per page on IEEE, 20 per page on Scopus). After this cutoff strategy, 1050 records from QS1 and 1500 from QS2 remained, having a total of 2550 articles.

After removing duplicates, 2220 records remained. A significant multi-stage filtering process was then applied, based on the criteria outlined in Table 3.2, resulting in the elimination of 784 records, narrowing the selection down to 1436 records. The remaining publications then underwent relevance screening.

The title review phase led to the removal of 1142 publications, an inspection conducted primarily manually, with a large language model (Google Gemini⁶) used to suggest *include/exclude* labels and a brief rationale (based on the research questions); the researcher reviewed every suggestion and always made the final inclusion decisions, and the subsequent abstract and full-text screening was performed entirely manually. Following this, the abstract reading phase further narrowed the selection, resulting in the exclusion of 214 additional articles. After this phase, 21 articles were removed either because no full text was accessible in Portable Document Format (PDF) or because, after reviewing the complete text, they were found not to be relevant. In summary, this rigorous filtering yielded a core set of 59 publications.

Finally, 16 additional studies were included via snowballing to mitigate limitations in database coverage, resulting in a final corpus of 75 publications. These additions did not pass through the screening pipeline of Table 3.2: they were selected directly, by their relevance to the research questions, which is what allowed foundational work predating the publication-year criterion to enter the corpus. The snowballing step followed references wherever they were indexed, which is how publications from venues outside the two queried databases, such as the ACM Digital Library⁷, entered the corpus. Three aspects of this process are acknowledged as limitations. First, the relevance-ranked cutoff depends on the databases' own ordering and is mitigated, but not eliminated, by the snowballing. Second, the 2010 publication-year criterion proved strict for the foundations of the field: half of the snowballed additions predate it, which indicates that the systematic search alone would have missed foundational work; the snowballing recovered it, but a broader temporal window would have surfaced it directly. Third, all screening decisions were

⁶Google Gemini, at <https://gemini.google.com/>

⁷The ACM Digital Library, at <https://dl.acm.org/>

made by a single reviewer, which is itself acknowledged as a limitation of the selection process. The restriction of the corpus to articles, conference papers, and books (criterion 5 in Table 3.2) likewise reflects a deliberate preference for archival, peer-reviewed publication types over theses and technical reports.

Table 3.2: Screening Criteria for Study Selection

Phase	Inclusion / Exclusion Criteria
1	Search engine result obtained from a query
2	Full-text access available
3	Published in English
4	Published in 2010 or after
5	Article, Conference Paper, or Book
6	Within Computer Science and related fields (e.g., Operations Research / Decision Sciences)
7	Not duplicated
8	Not a retracted publication
9	Not a work-in-progress publication
10	Not summaries of workshops, seminars, lectures, or tutorials

The complete identification and filtering pipeline, which resulted in a final corpus of 75 articles, is illustrated in Fig. 3.1.

3.1.4 Review and Analysis

The distribution of these publications by type is presented in Table 3.3, showing a balance between articles and conference papers. Moreover, Fig. 3.2 illustrates the distribution of the release year of the selected records.

Table 3.3: Number of reviewed publications by type and research topic.

Type	Visualization	Optimization	Total
Article	9	25	34
Conference Paper	19	20	39
Book	1	1	2
Total	29	46	75

The collected publications present some variety, but there are some patterns when comparing visualization and optimization. Regarding Visualization, the majority of the works focus on schedule views with interaction, often supporting editing and feedback through specific techniques. These approaches are frequently combined with spatial context when location and layout matter. Some papers also give importance to scalability and readability through aggregation and interaction design. Overall, the literature on this topic separates into 2D charts, spatial models with time, and performance and usability.

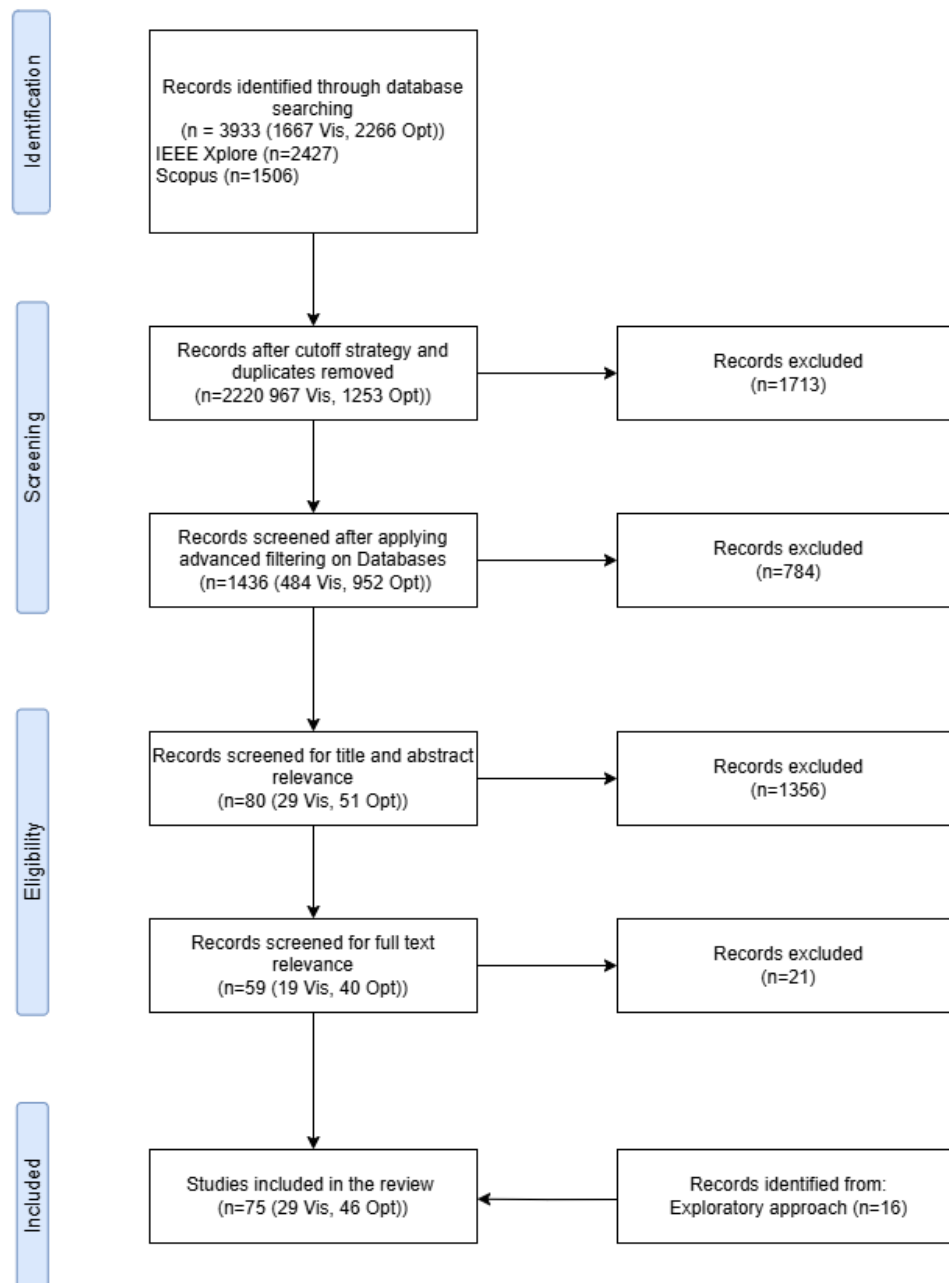


Figure 3.1: Identification and filtering pipeline for the literature review.

In optimization, the main topic is the Resource-Constrained Project Scheduling Problem (RCPSP) and its variants with multiple goals. Typically, objectives include reducing makespan and handling trade-offs between time and cost. Metaheuristics and hybrid methods dominate the spectrum, with strategies that combine global search with local improvement using exact or constraint techniques. Frequently, as models become complex, scalability is maintained using heuristics rather than exact methods. Moreover, some works discuss uncertainty and iterative replanning. Many evaluations rely on standard benchmarks or small real case studies.

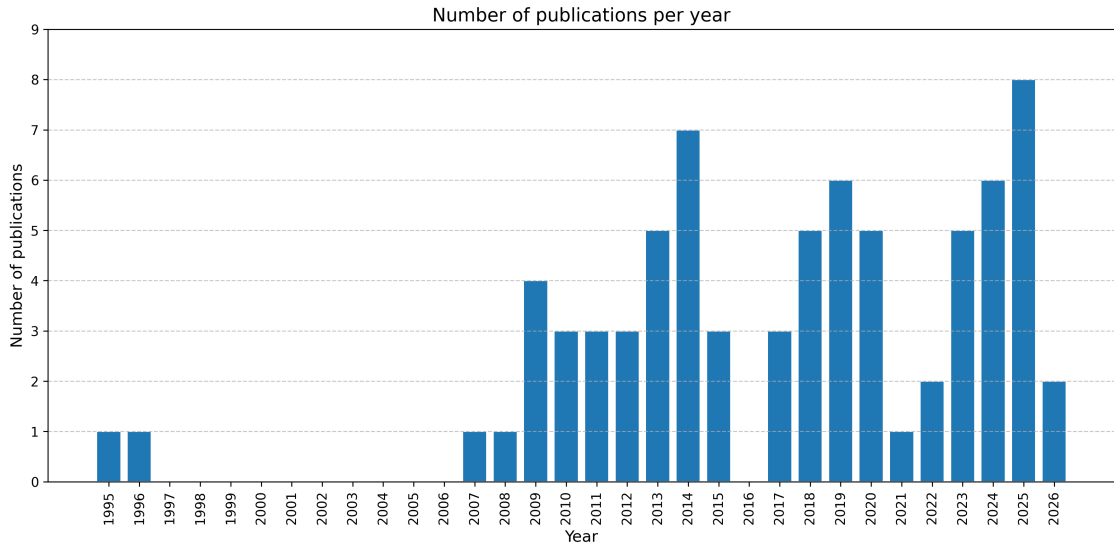


Figure 3.2: Publication year distribution of selected records.

The following sections present the state of the art on the topics discussed before (Visualization and Optimization). Section 3.2 presents the techniques found in visualization, while Section 3.3 discusses optimization methods for scheduling.

3.2 Visualization

Maintenance planning is not only a scheduling problem, as it is also essential for planners to explore and edit schedules under various constraints that interact with each other. So, this section reviews visualization techniques in order to support complex planning strategies.

Nevertheless, tools for planning are often not only about information visualization, but also about how to integrate computation. Visual analytics highlights the combination of visual interfaces and analysis, as a good visual interface facilitates reasoning when inspecting a schedule and exploring alternatives. Keim et al. (2008) describe a mantra: *Analyze First, Show the Important - Zoom, Filter and Analyze Further - Details on Demand*, which reflects the necessity for computation before and after the interaction when data becomes complex. This also aligns with Shneiderman's idea for visual interfaces: *overview first, zoom and filter, then details on demand* (Shneiderman, 1996).

The rest of this section will discuss visualization techniques that can be applied to project scheduling, starting in Section 3.2.1 with time-oriented visualizations and their associated scalability methods. After, Section 3.2.2 will approach dependencies and the visualization of constraints, discussing graph views and formal models to clarify the meaning of constraints. Moreover, Section 3.2.3 reviews spatial and spatiotemporal views, and then the focus will be on interaction and visual analytics, including coordinated views in Section 3.2.4. Section 3.2.5 joins interfaces and visual languages for constraint specification. Finally, Section 3.2.6 synthesizes the analysis from previous sections, summarizing the essential points and aiming to identify gaps in the literature.

3.2.1 Time-Oriented Visualizations

Initially, time-oriented representations were used for planning because schedules are defined over time, as the majority of activities occur as intervals or events, and coordination between activities is expressed through temporal relations. Nevertheless, representing time visually also has its challenges. Aigner et al. (2011) highlight that time-oriented data is influenced by several aspects, such as temporal granularity, chosen time primitives, whether time is fixed or uncertain, and the overall temporal structure. So, all of these points influence what can be shown and how users can understand it.

3.2.1.1 Gantt charts and interactive timelines

Despite many alternatives, Gantt charts are commonly used for scheduling (Aigner et al., 2011; Sakin & Isaacs, 2024). Sakin and Isaacs (2024) identify that interactive Gantt systems regularly support a good set of user tasks like: getting overviews, inspecting attributes from events, specifying and navigating time windows, and searching through schedules. The standard components of a Gantt chart, including tracks and dependencies, are illustrated in Fig. 3.3. Gantt charts are useful, but many of the tasks can become expensive or unreliable as the schedule size expands and attribute dimensions increase.

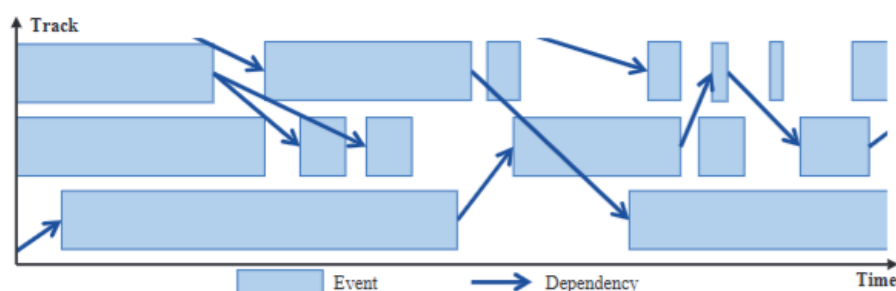


Figure 3.3: Example of a Gantt chart. Taken from (Sakin & Isaacs, 2024)

The problem is that scalability mechanisms are not trivial afterthoughts but crucial interaction design choices. Cockburn et al. (2009) present three major families of multiscale interfaces, namely overview+detail, zooming, and focus+context, which are frequently used to keep users

oriented while accessing detail, even though with known trade-offs that depend not only on the design choice but on the task. Practically, this translates to a concise set of techniques, such as hierarchical aggregation (collapsing/expanding), filtering and search, focus+context views, zooming, and attribute overlays like colors or labels, often combined with smooth navigation.

Regarding large-scale examples, Luz et al. (2009) propose Chronos, a system similar to Gantt designed for large schedules. Chronos integrates hierarchy directly into navigation, instead of this being a secondary feature, as users can fold or unfold tasks and adjust the level of granularity, which means that it supports both overview and drill down, instead of committing to a fixed scaling. This strategy directly addresses the problem of excessive two-dimensional scrolling, which often leads to context loss in extensive timelines. Through the metaphor of temporal mosaics, activities are summarized in dense, colored blocks. Luz et al. empirically show that this representation outperforms Gantt charts for temporal inference tasks, particularly when a logical hierarchy is defined.

From a different perspective, Jo et al. (2014) also target the scalability problem, with the introduction of LiveGantt interface, as seen in Fig. 3.4. This approach proposes a lens metaphor that provides semantic zooming between an abstract overview of the schedule and a detailed view. The objective is to balance both context and local structure when the schedule size grows considerably and becomes hard to manage (Jo et al., 2014; Tominski et al., 2014). The effectiveness of this approach was validated in a case study with a semiconductor factory (over 3,000 tasks), where users identified inefficiencies and reduced the makespan by 11% after reallocating only 15 tasks. However, the introduction of these aggregation layers may create new usability challenges in the detailed comparison of data (Jo et al., 2014).

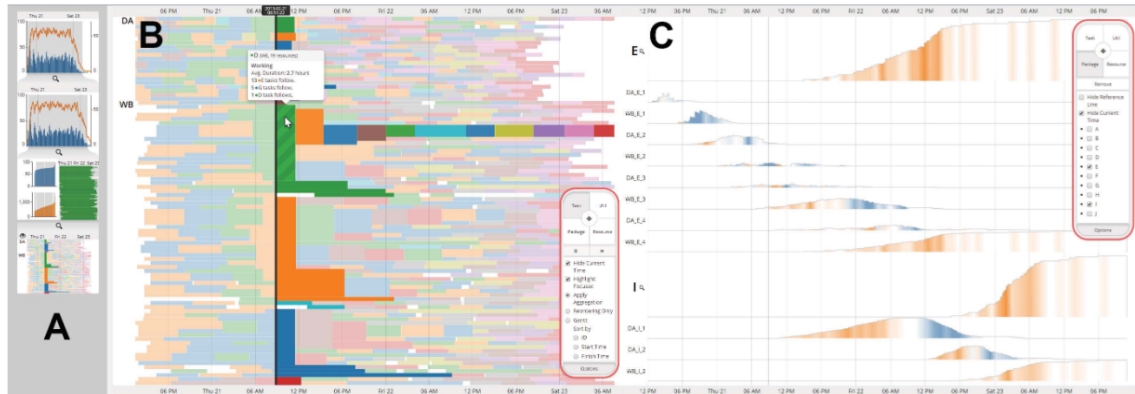


Figure 3.4: LiveGantt interface, combining aggregation, interaction history, and detail inspection. Taken from (Jo et al., 2014)

These two different approaches (Chronos and LiveGantt) illustrate a coherent message: interactive timelines are still commonly used for scheduling, however, only when combined with design choices that validate hierarchy, navigation, and inspection in large-scale schedules.

Despite these advances, views that only have time taken into account do not necessarily show why constraints induce conflicts or how dependencies shape feasibility. This limitation motivates

other types of representation of dependencies and constraint visualizations, which will be discussed in Section 3.2.2.

3.2.1.2 Alternative time representations

It was previously discussed that Gantt charts can be used effectively for scheduling. However, they are not the only way to encode time in planning, and other approaches are often discussed to clarify specific features that are difficult to view in a regular Gantt chart, such as the topology of precedence relations or the dynamics of repeated units. Undozerov (2022) states that while Gantt charts provide a direct view of temporal dependencies, network diagrams better reveal the topology of relations. In order to merge these perspectives and take advantage of both, the author proposes the spring chart technique, based on a spring metaphor, where the precedence relations are represented by the geometry of the spring, helping to communicate the dependency structure in relation to time as seen in Fig. 3.5. This perspective is beneficial when relationship patterns, such as dense precedence structures, are the primary analytical target. Despite its aim to make relational patterns more legible, it also lacks explicit mechanisms for connecting constraint logics to time placement.

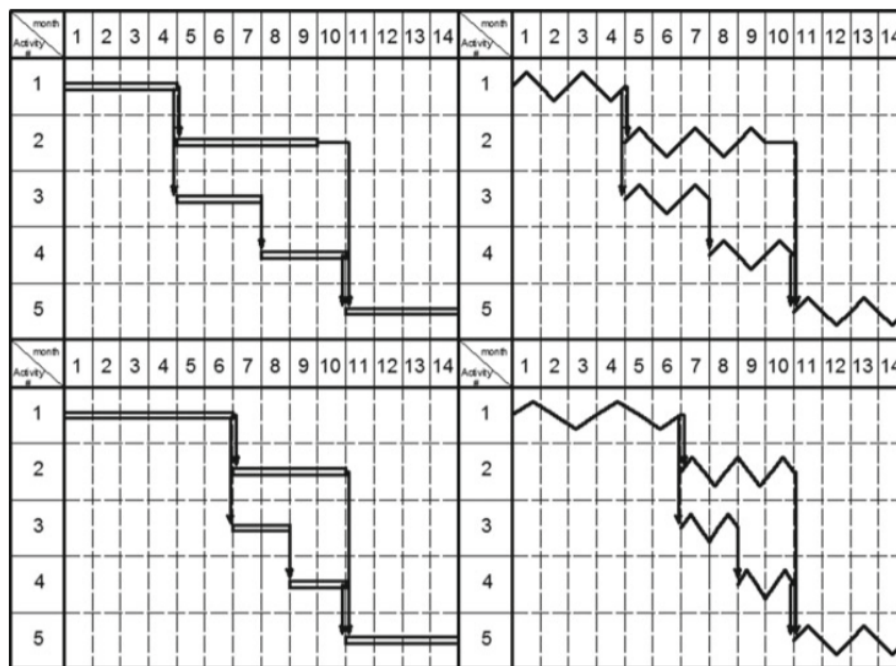


Figure 3.5: Example of the Spring chart technique compared with a Gantt chart for two schedule states. Taken from (Undozerov, 2022)

Another family of alternatives focuses on repetitive scheduling, where the main challenge is the flow of crews or activities through recurrent units, instead of the isolated start or end of individual activities. Hegazy et al. (2020) emphasize that Line of Balance (LOB) and Flowline diagrams are widely used for repetitive schedules, and they submit a visualization approach that balances duration and distance representation in order to improve interpretability for those types

of problems. This approach aims for a good domain fit because temporal aspects are maintained, at the same time that representation emphasizes repetition, pace, and continuity constraints that are usually lacking in standard Gantt visualizations. However, while these methods grant a strong domain fit for linear projects, their applicability is generally narrower than that of general-purpose scheduling representations.

3.2.2 Dependency and Constraint Visualizations

While timeline-based views inspect when activities occur, they may hide why a schedule is constrained. Network representations such as Program Evaluation and Review Technique (PERT) or Precedence Diagramming Method (PDM) diagrams expand the structure by explicating activities and their directed constraints. This structural view of precedence relations is used for inspecting patterns of connectivity (e.g., chains, forks, or joins, and feedback structures) and also to reason about how changes made locally can propagate across the whole schedule.

3.2.2.1 Precedence networks and dependency graphs

Network representations are notably effective for tackling structural questions, such as identifying long chains and bottlenecks, locating forks or joins, and anticipating how local edits may propagate globally. Industrial schedules are not only constrained by time dependencies, but also by resources like equipment, crews, and workspaces that create an implicit association between otherwise unconnected tasks. Ertek et al. (2015) expand the classical project network by incorporating resource nodes and connecting those nodes to their respective activities. This approach yields a graph that links resources and dependencies and facilitates inspection and explicitness as illustrated in Fig. 3.6. Furthermore, it describes layout procedures that focus on readability, such as the edge crossings reduction, ensuring that these links remain interpretable at a real project scale. Also, it frames the graph as a support tool for verifying dependencies that appear before scheduling decisions. Nevertheless, while this approach identifies resource-critical activities, it has been primarily applied to single networks, and multi-project visualization remains a documented challenge.

Even though precedence is already represented, richer dependencies can introduce a non-intuitive criticality behavior. Nisar and Suzuki (2015) developed a critical activity analysis using the PDM with other relations, including finish-to-start and time lags, which separate different behaviors and emphasize irregular cases in which reducing the critical activity can increase the project duration overall. Moreover, they categorize these behaviors as normal, reverse, or neutral, providing a framework in order to understand these effects. To address these effects, visual diagnostics centered on dependencies are used, as users need to check not only which activities are critical, but also the specific relations among them.

Even with these layout steps, graphs become increasingly difficult to read as the number of nodes grows, so more advanced methods like grouping or filtering are needed to keep them clear. Because dependency graphs reinforce structure rather than absolute time placement, they are often

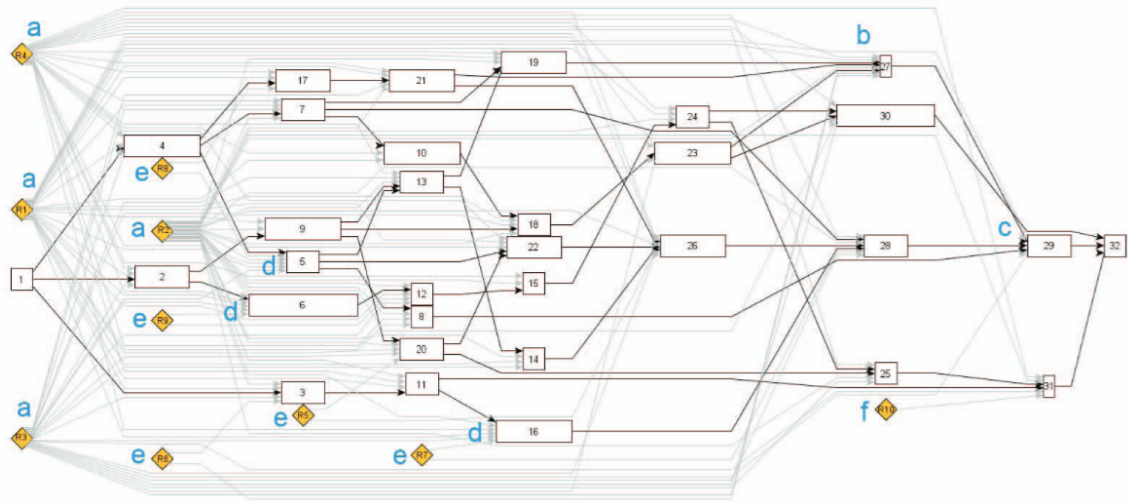


Figure 3.6: Project network augmented with explicit resource nodes using a hierarchical layout to expose resource dependencies. Taken from (Ertek et al., 2015)

effective when combined with a temporal view. Multiple coordinated views can be joined to a network diagram, so that features from each view are reflected in the other through techniques such as brushing and linking. This approach supports rapid inspection between graphs and timelines (Roberts, 2007).

3.2.2.2 Formal models for constraints

Whereas graph visualizations articulate the structure of constraints, the meaning of complex dependencies is difficult to validate informally, especially when relations other than finish-to-start, or lags, or conditional enabling are present. It has been shown that models using Petri nets provide an executable description where constraints have operational meaning to enable consistency verifications (Jamnuch & Vatanawood, 2019; Tsinarakis, 2018).

Tsinarakis (2018) models the four standard task dependency types, shown in Fig. 3.7, including finish-to-start (FS), start-to-start (SS), finish-to-finish (FF), and start-to-finish (SF), along with their lagged variants, using Petri nets with arc extensions, such as inhibitor or test mechanisms. These extensions enable the simulation of complicated interrelated tasks and the comparison of outcomes with standard schedule representations.

In a similar vein, Jamnuch and Vatanawood (2019) propose transformation rules for converting activity network diagrams into timed Petri nets. This technique reduces the manual effort required to check whether a complex network is correct and, at the same time, enables verification via timed semantics. This process, often supported by verification software such as Colored Petri Nets (CPN) Tools, allows for the detection of differences between intended and actual execution behavior. In reality, these conventions help detect discrepancies and explain why a set of dependencies produces a specific execution behavior. In this case, the main trade-off is usability: even with formal semantics that improve verifiability, the representation is not as accessible as in the timeline view. In practice, Petri net models have a steep learning curve and are not well suited

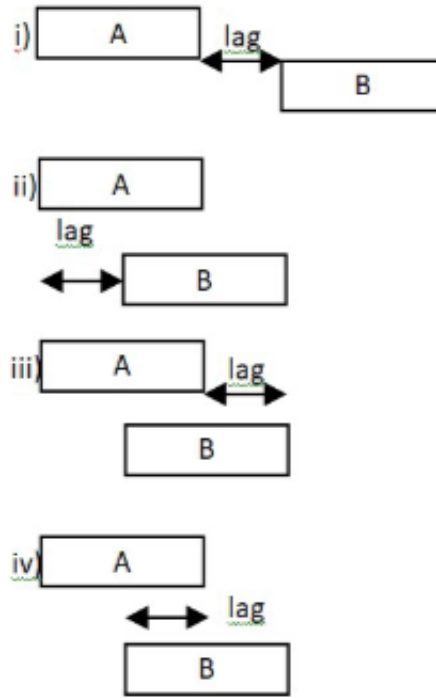


Figure 3.7: Standard task dependency types (FS, SS, FF, SF) as scheduling primitives that can be mapped to formal constraint models. Taken from (Tsinarakis, 2018)

to everyday planning by non-specialists. So, systems regularly leverage formal results to reveal visual summaries, rather than expecting planners to manage Petri nets directly.

A practical decision rule emerges for large-scale planning. When the central purpose is to clarify infeasibility and resource coupling for planners, an augmented dependency graph offers the most actionable solution. In contrast, for exacting semantic verification of complex logic, formal models should function as back-end engines, with their results presented through visual summaries accessible to planners. Together, these approaches make precedence and resource coupling more explicit. Nonetheless, spatial restrictions are also essential in maintenance planning, which motivates the spatiotemporal views discussed in Section 3.2.3.

3.2.3 Spatiotemporal and Spatial Visualizations

Views focused on time and dependencies explain when tasks occur and why they are ordered, but they are not space aware. Workspace clashes, access restrictions, and safety zones are some examples of spatial constraints, and spatiotemporal visualizations address this problem by joining temporal structure with spatial abstractions so that these dependencies become visible explicitly.

3.2.3.1 4D scheduling visualization

In the industrial domain, four-dimensional (4D) scheduling, which often integrates computer-aided design (CAD) and building information modeling (BIM), is often represented as an animation of

3D objects through time. Nevertheless, typical 4D animations often make inspection difficult because durations are not visually explicit, upcoming work is not simultaneously visible, and workspace conflicts are not always evident, which limits the overall schedule overview and makes it harder to spot overlaps. Ivson et al. (2018) address this problem with CasCADE, illustrated in Fig. 3.8, a framework that maps time as a spatial dimension in a 3D view representation. The objective is to make task sequencing and spatiotemporal overlap evident in a single view, while still keeping the spatial context of the BIM geometry.

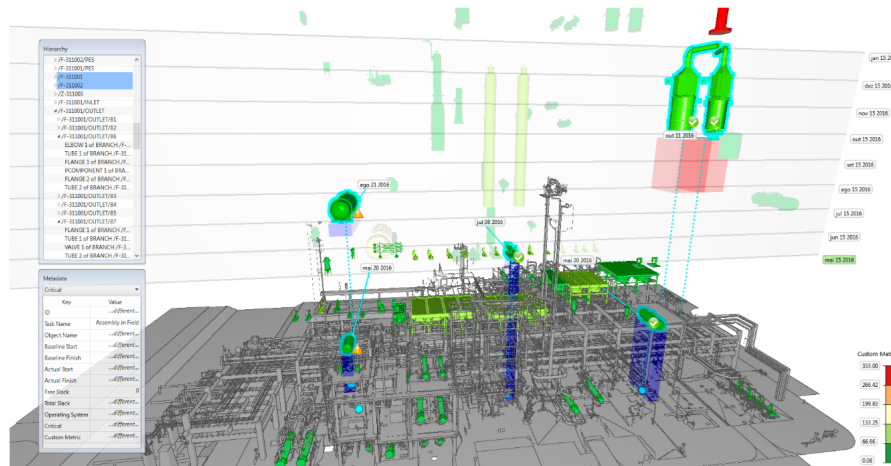


Figure 3.8: CasCADE 4D visualization concept, mapping time to a spatial axis to support overlap inspection in construction planning. Taken from (Ivson et al., 2018)

This system connects various visual devices, such as color coding, transparency, and focus+context interaction, and it supports the inspection of durations, dependencies, and workspace disputes during schedule review (Ivson et al., 2018). The evaluation of the system was conducted in a scenario from an Oil and Gas facility, where the use of CasCADE enabled the detection of uncertainties and conflicts in the schedule, related to missing information and spatial obstructions. However, this study also demonstrated that depth complexity and occlusion can accumulate rapidly in 4D views, making mitigation techniques such as transparency essential for usability.

Chang et al. (2017) present another work that uses BIM visualization in scheduling. They combine a critical path model that incorporates risk with a 3D BIM environment. With this, it allows planners to estimate durations more effectively than using a 2D approach. Within this field, geometry primarily serves as contextual evidence to enhance duration estimates and facilitate comparisons of risk scenarios during schedule development, rather than for clash detection. While the essential focus is to estimate duration under risk instead of geometric conflict visualization, this study also gives the example of how spatial context is used as an aid in schedule reasoning.

The use of 4D views is most appropriate when reliable geometric data is available at the necessary level of detail and when planners must evaluate local spatial feasibility, particularly the safe concurrent presence of multiple interventions within the same physical volume during overlapping time periods.

3.2.3.2 Spatial factors and occupancy

Not only is the goal to show construction progress, but various approaches also treat spatial constraints as the main objective. Semenov et al. (2013) propose a method in which traditional precedence or resource restrictions are enriched with spatial constraints that cannot be reduced to standard resource limits. This technique alternates between scheduling for spatial constraints and standard planning using visual modeling for spatial validation. After this, the spatial conflicts that emerge from the process result in an iterative repair plan that modifies links. The applicability of these workflows depends on exact mappings from tasks to spaces and on the quality of the underlying spatial model.

At the other end, occupancy and chronographic models have the goal to capture temporal and spatial constraints more compactly. Morin-Pepin and Francis (2018) work, illustrated in Fig. 3.9, proposes modeling concepts to classify operations and space occupations, like zoning by floors and sectors. Also, they define occupation types in order to compute a dynamic occupancy rate that considers space usage over time and congestion. The main distinction is between inclusive and exclusive occupation, which addresses cases where a task blocks adjacent work even if only using a sector partially. This helps planners reason about workspace and site flow without needing complex 3D analysis at every step. These abstractions scale more effectively than full 3D simulations and remain usable when detailed BIM or CAD coverage is incomplete.

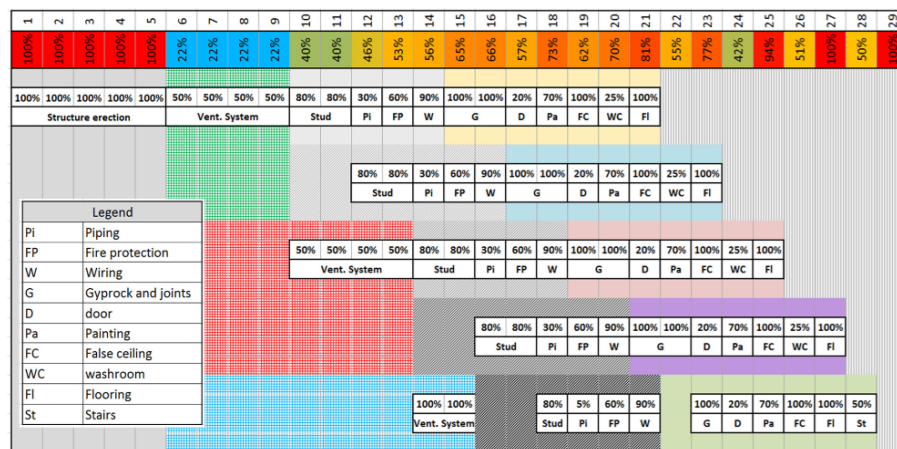


Figure 3.9: Chronographic schedule model to monitor site congestion via a dynamic occupancy rate over time. Taken from (Morin-Pepin & Francis, 2018)

Zone or occupancy abstractions are most suitable when planning emphasizes capacity and congestion instead of exact geometric interference, and when spatial data is primarily represented as defined areas or sectors. Collectively, these studies exhibit a fundamental trade-off: while rich four-dimensional geometry offers greater expressiveness, it is computationally expensive, whereas spatial abstractions are more scalable for handling specific spatial constraints that affect planning decisions.

3.2.3.3 Safety centric and spatial views

Rahman et al. (2018) propose a planning assessment pipeline in which alternative plans are generated from a network activity graph and subsequently integrated with motion planning to produce paths and safety policies for workers and equipment. Safety scores and heatmaps are calculated to visualize hazardous zones across both time and space. This approach conceptualizes safety as a property with explicit spatial outputs and informs planning decisions involving deadlines, resources, and cost. In the context of scheduling, this suggests that temporal encodings are insufficient when risk is governed by spatial proximity and routing. In such cases, visual representations can support interpretation by making risk drivers and hazardous regions explicit (Rahman et al., 2018). Nevertheless, moving these safety-focused systems relies heavily on specific map layers, such as hazard maps and availability graphs, and on how well these route choices can be linked to existing schedules.

The reviewed literature suggests that the selection of spatial representation should be guided by pragmatic criteria. Rich 4D views are highlighted for local feasibility and interference checks when detailed geometry is available (Ivson et al., 2018). Occupancy abstractions are more appropriate for monitoring global congestion and site flow in large-scale schedules with partial Building Information Modeling (BIM) coverage (Morin-Pepin & Francis, 2018). Safety-centric layers are necessary when the primary planning objective is to decrease exposure and optimize routing in hazardous environments (Rahman et al., 2018).

Although these spatiotemporal constraints are effective, they also increase interaction complexity. The following Section 3.2.4 examines coordinated views that integrate temporal, structural, and spatial dimensions within one unified interface.

3.2.4 Visual Analytics and Interactive Planning

While visualizations support constraint reasoning, operational planning usually requires more than reading. Planners should relate schedules to performance indicators and identify anomalies at the same time that they refine decisions with algorithmic support. Visual analytics defines this as the combination of analysis methods and interactive interfaces to facilitate logical reasoning and decision-making (Keim et al., 2008). Regarding scheduling, this typically materializes as an iterative loop: *generate (or update) a plan → inspect with visual evidence and key performance indicators (KPIs) → edit or repair → recompute and compare outcomes*.

3.2.4.1 Dashboards and multiple views

Sun et al. (2020) present a visual system for production scheduling in factories, targeting both efficiency and rapid response to incidents. This context underscores the need for coordination, as planning often involves multiple factories and products. Planners must balance objectives, including delay, cost, inventory levels, and capacity smoothing. PlanningVis interface, as shown in Fig. 3.10, is composed of coordinated views across three different levels of detail. The first is a plan overview for the global context, the second is a product view to visualize various properties

of individual products, and the last is a production detail view to display product dependencies and daily production details in related factories. PlanningVis was validated through two industry case studies using real production-planning data (e.g., a 30-day plan with 1038 products) and expert feedback sessions. Moreover, these views are complemented by interactive operations such as filtering, searching, and linked selection (Sun et al., 2020).

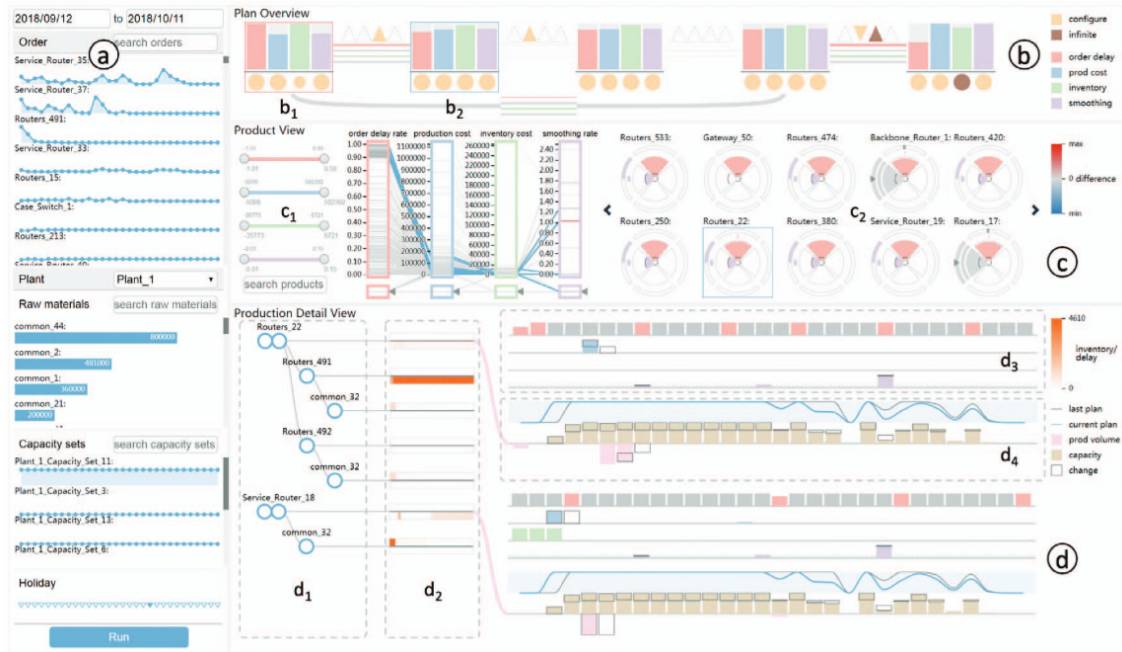


Figure 3.10: PlanningVis interface with coordinated views for plan overview, product level inspection, and detailed production analysis. Taken from (Sun et al., 2020)

Overall, this organization aligns with the broader pattern of coordinated and multiple views from Roberts (2007), where various views are shown, and they are interactive to support exploration. Other examples seen before, like LiveGantt, provide further evidence that this design remains feasible under large schedules (Jo et al., 2014). Ultimately, coordinated views are remarkably effective for exploratory diagnosis, as they facilitate the identification of irregularities across temporal and asset dimensions, enable the determination of responsible entities, and support detailed investigation of specific causes while keeping a broad overview.

3.2.4.2 What-if analysis and scenario comparison

Beyond the exploration of a single schedule only, scheduling tools should support what-if reasoning, allowing the planner to explore different choices and assess trade-offs. The example from before, PlanningVis, supports this what-if analysis and quick schedules comparison through an overview that stores the history of optimization, and allows users to inspect differences. Moreover, it proposes a glyph representation that transmits performance indicators and the differences

among plans (Sun et al., 2020). This design highlights a critical requirement for decision support systems: alternatives should be easily comparable for initial screening and allow for in-depth inspection to support detailed justification.

In complex planning scenarios, what-if analysis serves as the principal mechanism that transforms visualization tools into effective decision-support systems. This approach links manual revisions and algorithmic re-optimization to quantifiable impacts (KPIs), thereby supplying the evidence required to justify planning decisions to stakeholders.

3.2.4.3 Interactive planning support

Various systems aim to close the gap between visualization and scheduling by integrating interactive exploration with automated reasoning in a bidirectional manner. Schneider and Aigner (2011) propose A-Plan, which integrates planning and visualization by merging user interaction with an optimization algorithm to strengthen the schedule. The authors report that primary operations can be performed substantially faster than in existing workflows, showing the operational benefit of integrating interaction with optimization. A-Plan was evaluated with five participants (three current users and two IT experts) performing seven tasks; for instance, booking a customer maintenance task took about 4 minutes in the current system versus a few seconds in A-Plan (Schneider & Aigner, 2011). This method is relevant when approaching the real world, where scheduling decisions require iterative refinement instead of a single optimization.

Kim et al. (2013) propose another complementary example, Cobi, that incorporates community-sourced constraints and preferences. This mixed tool enables planners to easily spot conflicts and receive recommended swaps, which allows them to preview the impact of changes before committing. The system's effectiveness was demonstrated through its deployment at Computer-Human Interaction (CHI) 2013, where the organizers used the tool for 42 days to resolve 168 conflicts, managing 8651 author preferences and 1722 associate chair affinities. The proposed system demonstrates the feasibility of incorporating optimizer-in-the-loop support, in which organizers reconcile explicit constraints alongside implicit considerations while retaining control over the final decision-making process (Kim et al., 2013).

The reviewed systems show two complementary interaction schemes. While coordinated views support exploration and diagnosis and are effective for finding anomalies and comparing alternative plans using key performance indicators, optimizer-in-the-loop approaches facilitate mixed-initiative repair. They are suitable for efficiently resolving conflicts and guiding user edits with previews, while sustaining user control. Table 3.4 summarizes the key strengths and requirements of these two patterns, highlighting their specific roles in the planning process.

In the reviewed systems, the prevention of user errors relies more on the feedback mechanisms surrounding edits than on the notation itself. Effective interfaces typically integrate the following elements:

- **Immediate violation feedback:** highlight the affected entities, such as tasks, rooms, or resources, at the time of an edit and specify the rule being violated, rather than providing a

generic “infeasible” status (Thomas et al., 2010).

- **Safe what-if previews:** enable tentative moves or swaps and display their downstream consequences prior to commitment, thereby reducing cascading errors and facilitating negotiation (Kim et al., 2013).
- **Guided repair or relaxation:** when infeasibility remains, recommend minimally disruptive repairs, such as swaps, or explicitly support controlled relaxation of selected constraints, thereby establishing revision as a primary operation (Kim et al., 2013; Thomas et al., 2010).

Table 3.4: Comparison of visual analytics patterns for interactive planning.

Pattern	Strengths	Costs
Coordinated views	Multi-level diagnosis; Comparison of KPI's; identifying anomalies across dimensions.	KPIs and cross-links must be well defined; risk of cognitive overload without filtering.
Optimizer-in-the-loop	Fast conflict resolution; recommendation of edits; consequence preview.	Requires explicit constraint models and preview mechanisms for decisions making.

In the CERN planning context, where feasibility depends on the interaction among several constraints, a hybrid loop is the most promising approach. This approach couples conflict diagnosis on an enriched schedule view with optimizer-in-the-loop support that proposes repair actions and previews their consequences. This rationale underpins the solution presented in Chapter 4, which provides a visualization environment that makes conflicts interpretable and supports iterative, justifiable plan refinement, rather than relying on single optimization.

3.2.5 Constraint Specification Interfaces

Because constraint-rich planning depends on how constraints are elicited and encoded, it is essential to examine the interaction approaches used to specify them. This subsection summarizes interface approaches to constraint specification, beginning with visual languages that provide structured, semi-formal representations.

Consider a typical set of requirements in an industrial setting: spatial exclusivity (e.g., only one task at a time in Zone Z), safety prerequisites (e.g., Task B requires a specific safety permit), and temporal windows (e.g., tasks must be completed within defined limits). While a form-based editor would present these as fields to complete, a visual tool would display them as visual primitives, such as objects for zone limits or lines indicating task order. Whenever a constraint is violated, the system would display a warning and suggest corrections on that specific task. On the other hand, another system might prompt the user to relax constraints, for example, by extending time limits (Kim et al., 2013; Thomas et al., 2010).

3.2.5.1 Visual languages for constraint specification

In scheduling, constraints are rarely static, as they must be introduced, negotiated, and revised. Thus, constraint specification becomes an analytical process instead of a simple data entry. One way to discuss visual interfaces is through principles for visual notations. Moody (2009) describes a set of principles for evaluating visual notations by how well they communicate. These principles include semantic transparency (suggesting meaning through symbols), graphic economy (avoiding excessive symbol variety), and complexity management (preventing complicated constraints from collapsing into visual noise). Together, these principles provide a base for constructing notations that remain communicative even when constraints become heterogeneous and interdependent.

In this context, the key is the level of detail when visualizing constraints. Interfaces typically span a spectrum:

- **Implicit specification (parameter editing):** Constraints are encoded indirectly by setting parameters or inputs. While this reduces the learning burden and supports fast data entry, it can hide the underlying structure of the constraint network, making conflict diagnosis more challenging.
- **Explicit specification (visual primitives):** Constraints are manipulated as first-class objects, exposing their types and relationships directly. This improves traceability and debugging, but it risks notational overload.

Hegazy et al. (2020) reinforce this by showing that expressiveness must be balanced with abstraction to prevent diagrams from becoming visually crowded as more logic is presented.

3.2.5.2 Constraint editing workflows and usability

The editing workflow defines the usability of these interfaces, how users detect violations, understand constraints, and repair errors. Cobi, for example, demonstrates this by letting users preview the consequences of their edits and receive recommendations for changes (Kim et al., 2013). This design recognizes that automated scheduling often fails to grasp the meaning of constraints that are difficult to formalize. By combining conflict visibility, what-if feedback, and guided repair, the workflow allows users to reconcile encoded rules with tacit knowledge before committing.

In contexts such as timetabling, visual interfaces assist users in identifying conflicts and directing the search toward feasible solutions (Thomas et al., 2010). These interfaces allow users to formalize constraints through dedicated menus and translation tools. If a problem is over-constrained, the workflow facilitates constraint relaxation, making rule revision an explicit step in achieving feasibility.

Table 3.5 summarizes these interface strategies, their strengths, and typical failure modes.

In the context of CERN shutdown planning, where constraints are both numerous and negotiable, the interface should facilitate authoring (expressing rules), explanation (clarifying violations), and revision (modifying constraints). Approaching constraint specification as an interactive

Table 3.5: Comparison of constraint specification interface strategies.

Strategy	Pros	Cons	Mitigation
Implicit	Fast entry; low learning overhead	Hidden structure; weak conflict explanation	Conflict highlighting; summaries; provenance views
Explicit	Traceability; debugging structure	Visual clutter; scale issues	Complexity management (aggregation, filtering) (Moody, 2009)
Mixed-initiative	Rapid repair; iterative refinement	Reduced transparency of recommendations; need for user trust	What-if preview; explanations; reversible edits (Kim et al., 2013)
Interactive steering	Over-constrained problems; negotiation	Endless trial-and-error	Guided relaxation; progress feedback (Thomas et al., 2010)

analysis task that achieves semantic transparency while managing complexity is key to ensuring system usability at scale (Moody, 2009).

3.2.6 Comparison of Approaches

Table 3.6 compares the visualization approaches discussed in this section.

It was shown that timeline representations continue to serve as the primary navigation framework, as planning is fundamentally linked to temporal relationships. As constraints become increasingly complex, these representations should be supplemented with structural models to clarify dependencies and propagation effects, and with spatial methods to address site flow and hazardous areas. Visual analytics conceptualizes this process as an iterative loop that combines planners’ tacit knowledge with automated computation. A visual language is effective only if the interface directs users toward valid configurations, prevents violations, and transparently communicates the consequences of decisions through mixed-initiative feedback.

The analysis shows that no single representation is adequate. An effective baseline combines a time-oriented perspective for global structure with dependency views to explain conflicts, supported by multiscale interaction and coordinated selection. Spatial reasoning, along with zone-based abstractions, delivers a scalable approach to monitoring congestion across activities. Since shutdown plans are subject to negotiation and incident response, a mixed-initiative framework that unites recommendations and consequence previews represents the most effective approach for incremental refinement.

Despite these foundations, the literature identifies substantial gaps in the CERN context: First, most systems assume clean data links, but in practice, it is often hard to fully map tasks, resources, zones, and safety layers. Second, while current tools show where issues happen, planners still do not get clear explanations for why they occur or what the simplest fixes are. Third, it is still

Table 3.6: Summary of visualization approaches.

Category	Literature	Strength	Role
Temporal	Chronos, LiveGantt (Jo et al., 2014; Luz et al., 2009)	Large-scale timelines	Baseline navigation; requires hierarchical aggregation.
Structural	Graphs, Petri Nets (Ertek et al., 2015; Tsinarakis, 2018)	Dependency diagnosis	Explaining conflict logic via summaries.
Spatial	CasCADE, Chronographic zones (Ivson et al., 2018; Morin-Pepin & Francis, 2018)	Site congestion analysis	Global flow (zones) and local clash detection (4D).
Analytics	PlanningVis, A-Plan, Cobi (Kim et al., 2013; Schneider & Aigner, 2011; Sun et al., 2020)	What-if decision support	Integrated loop for iterative diagnosis and repair.

difficult to systematically compare compromises between time, safety, and resources in integrated environments. Finally, many interfaces do not support easy, step-by-step editing of constraints, which slows down work on over-constrained problems.

These points motivate the solution adopted in this work: a reusable abstraction that captures the recurring groups of activities characteristic of this domain (Section 2.2.1), an enriched timeline that brings dependencies and constraints onto a single coordinated surface, an explicit account of why a schedule is infeasible together with the edits that would repair it, and a mixed-initiative review in which planners accept, reject, and adjust the proposed schedule. This approach intends to make shutdown planning more transparent, more justifiable, and easier to improve step by step.

3.3 Optimization

Generally, scheduling is studied using the RCPSP, where the objective is to construct a feasible schedule that minimizes the project’s makespan. Usually, a project consists of a group of activities that cannot be interrupted, with known durations and limited resources (Karam & Lazarova-Molnar, 2013). If resource constraints are ignored, the problem reduces to precedence scheduling, that can be solved by computing the critical path (CPM) (Kelley & Walker, 1959; Malcolm et al., 1959). The addition of resource constraints is a key factor that makes this problem computationally hard (Artigues et al., 2026; Karam & Lazarova-Molnar, 2013).

The RCPSP has been investigated for decades, both in academia and in industry, because it is an NP-hard (non-deterministic polynomial-time hard) (Artigues et al., 2026; Karam & Lazarova-Molnar, 2013) problem and uncovers the fundamental trade-off between solution quality guarantees, scalability, and expressiveness when consolidating richer constraints (Artigues et al., 2026).

The classic problem is intentionally lean, therefore contributing to its role as an engaging benchmark and a learning hub for an extensive spectrum of optimization methods. Nevertheless, its assumptions are often too strict for real-world cases, consequently encouraging a broad ecosystem of variants (Artigues et al., 2026).

Among these, several address the calendar restrictions that pervade real projects, where work is suspended over nights, weekends, and holidays. The distinction between activities that are interrupted by such breaks and those that, for technical reasons, run through them is long established (Franck et al., 2001), and constraint-programming procedures with specialized calendar propagators have since been developed for the resource-constrained project scheduling problem with calendars (Kreter et al., 2015).

Alongside this rich problem landscape, the field is supported by shared evaluation resources: standard benchmark libraries, as well as open competitions and testbeds that offer standardized instances, objective definitions, and evaluation protocols. These resources enable reproducible comparisons across algorithmic families. For example, the Integrated Healthcare Timetabling Competition (IHTC-2024⁸) introduced an integrated timetabling problem with public datasets and a ranking procedure under fixed computing budgets (Ceschia et al., 2025). While IHTC addresses healthcare timetabling rather than project maintenance, it serves as a relevant testbed for constraint-rich, integrated scheduling problems. This allows scalability and solution performance to be evaluated within a common experimental framework.

The rest of this section will discuss optimization approaches for project scheduling, starting from exact and constraint-based methods in Section 3.3.1, moving to heuristics, metaheuristics and hybrid approaches that combine the advantages of both methods in Section 3.3.2. It then reviews extensions that reflect real-world scenarios, accounting for multi-objective formulations and handling uncertainty, and learning-based techniques in Section 3.3.3. Finally, Section 3.3.4 synthesizes the discussions from the previous sections, summarizing the essential points, and identifying gaps in the existing literature.

3.3.1 Exact Methods

Exact methods aim to produce schedules with reliable guarantees, such as proven optimality or close lower or upper bounds for a given goal. In research, these methods have two main roles. They solve smaller problems completely to help with decision-making and benchmarking. They also work as strong solvers in larger strategies, such as rolling-horizon, relax-and-solve, or decomposition methods, where only part of the problem is solved exactly.

The main objective in project scheduling is to compute schedules that are optimal with respect to a certain objective (Artigues et al., 2026; Karam & Lazarova-Molnar, 2013). Exact methods are interesting as they provide optimal schedules, which are essential for decision support and for testing more scalable approaches. Still, as the problem complexity increases, both the size and combinatorial nature of the search space immediately challenge exact solving. This section

⁸More information at <https://ihtc2024.github.io/>

discusses two exact families and emphasizes strategies used in the literature review to expand them into real-world, larger-scale instances.

To support method selection from a decision-making perspective, Table 3.7 summarizes the main practical trade-offs between mixed integer linear programming (MILP) and constraint programming (CP) for constraint-rich scheduling problems.

Table 3.7: Comparison of MILP and CP.

Aspect	MILP	CP
Modeling focus	Emphasizes cost optimization within a linear structure.	Prioritizes feasibility and supports complex rules through global constraints.
Time and logic	Frequently requires discretization or linearization of time and logic.	Natively handles intervals and precedences, enabling expressive temporal rules.
Resources	Efficient for linear and aggregate capacity models.	Well-suited for cumulative resources, calendars, and complex resource logic.
Multi-objective	Supports straightforward implementation of weighted objectives and trade-offs.	Capable of handling multiple objectives, though these may require additional tuning or modeling decisions.
Proof/optimality	Provides bounds and certificates when solved to optimality.	Can be effective, although proving optimality may be more challenging in certain cases.
Re-planning	Supports warm starts, though reformulations can be computationally expensive.	Often effective at repair and adjustment through constraint propagation.

3.3.1.1 Mixed Integer Linear Programming (MILP)

MILP remains a classical exact paradigm for project scheduling models with precedence relations, time, and resource constraints, as it uses powerful mathematical techniques to combine variables within linear equations, in order to find the best solution under given constraints (Klein et al., 2024). In practice, MILP usually works best when the goal is to minimize costs that can be added up, the rules can be kept simple, and clear limits are needed to help make decisions. In sum, MILP formulations can become hard to scale as instance size increases and constraints require complex combinatorial reasoning. This often results in many binary variables and weaker relaxations, motivating other paradigms, such as CP, which rely on constraint propagation.

Lerch and Trautmann (2019) propose a lazy constraints strategy, with the purpose to enhance scalability by avoiding the enumeration of all constraints in advance. This model starts from a relaxed conceptualization, where violated constraints are added dynamically. Their approach relies on identifying minimal forbidden sets (MFS) from infeasible solutions, and adding only the corresponding cuts. This allows them to reduce the initial model size and raise performance

when compared to other existing MILP formulations. The computational study utilizes the J30 set of the Project Scheduling Problem Library (PSPLIB), comprising 480 instances with 30 activities each, and applies a 500-second time limit. The results demonstrate that MILP scalability improves significantly when the complete constraint set is too large to enumerate in advance.

Klein et al. (2024) introduce a compact continuous-time MILP model and achieve strong results on standard benchmarks. For example, on the J30 set with 480 instances, the best setup proves optimal in most cases (471 out of 480) and averages about 15 seconds per run under a 500-second limit. These results suggest that MILP can be very effective and provide certificates for structured instances.

Folco et al. (2023) introduce an Integer Linear Programming (ILP) technique for rolling stock maintenance scheduling that integrates both operational and logistical constraints to minimize total maintenance cost. Based on the results, their model efficiently works in industrial scenarios while having an acceptable computational runtime, concluding that linear modeling can be adapted to real-world cases. In their setup, they achieve short runtimes on generated instances with up to 200 lines and 200 days (mean runtimes $\approx 1\text{--}2$ seconds), showing that MILP is a reasonable approach when constraints can be written linearly, and the problem size is manageable.

Z. Ma et al. (2025) also propose a similar formulation, which proposes a coordinated optimization regarding traffic and task scheduling in construction. By combining vehicle tasking with safety and time constraints, their formulation aims to improve efficiency and, at the same time, ensure safe and feasible operation. This highlights MILP's strength in integrating operational constraints into one exact framework.

In sum, this method struggles when the problem becomes substantially larger or when its constraints require complex combinatorial thinking. Thus, constraint programming paradigms appeared in order to propagate constraints effectively and model scheduling directly.

3.3.1.2 Constraint Programming (CP)

Constraint Programming (CP) differentiates itself in scheduling because it allows for the representation of temporal relations and resource constraints more expressively, while also pruning the search space through constraint propagation. These models are more intuitive, making this a feasible method if the objective is to capture rich operational rules, rather than only linear constraints. Constraint Programming (CP) is most effective when the constraint set is extensive and rapid generation or repair of feasible schedules is required. However, CP regularly encounters difficulties in providing strong dual bounds, and establishing optimality can be computationally costly in large combinatorial problems.

Jung et al. (2025) provide an example of task scheduling in Maintenance, Repair, and Overhaul (MRO) operations, specifically for aircraft engines. They proposed a CP method that accounts for realistic constraints and multiple assemblies in maintenance scheduling. This technique was used to increase resource utilization and reduce total makespan, highlighting that CP is a good approach when incorporating rich constraints. They report high resource utilization in the presented case (e.g., 99.34%), illustrating CP's strength in capturing detailed operational rules.

Not only is CP applied to aircraft, but also to other fields like robotics. Behrens et al. (2019) propose a CP solver for simultaneous task allocation and motion scheduling. They introduce Ordered Visiting Constraints (OVC) to specify tasks with their spatiotemporal requirements and ordering constraints. Also, their model, implemented using Google's OR-tools, has obtained near-optimal plans quickly for typical problem sizes. They report typical instances with up to ~ 200 objects solved within minutes in their setting, supporting CP as an effective approach for richly constrained scheduling.

Similar to MILP, CP does not escape the scalability issue in large combinatorial instances. Hu et al. (2025) propose a CP approach with a rolling-horizon heuristic for repetitive project scheduling. They combine the advantages of CP with heuristics, and compare the results with both standard MILP and CP models. Their results reflect a common pattern observed in larger instances: a feasibility-first or phased strategy can preserve CP expressiveness while substantially reducing runtime, often achieving near-optimal quality much faster than a pure exact solver.

This line of work provides a natural transition to the upcoming section, where exact models meet heuristics and hybrid architectures, in order to retain modeling expressiveness while improving computational compliance.

3.3.2 Heuristics and Metaheuristics

As it was discussed in the last section, when models become more realistic (e.g., larger instances, additional constraints, more resources, multiple nodes), exact methods solely become impractical, which motivates the use of heuristics (specific for a particular domain) and metaheuristic (generalized, and can be applied to several problems) approaches for these NP-hard problems. These methods have the same objective: to obtain excellent results in a reasonable time, exploring the trade-off between optimality and time (Artigues et al., 2026; Karam & Lazarova-Molnar, 2013).

3.3.2.1 Priority-rule heuristics and local search

In many operational settings, planners need feasible schedules quickly and under evolving conditions. This motivates heuristic approaches when solution quality under a time budget is more relevant than proving optimality (Hartmann, 2021).

Priority-rule heuristics start from scratch and build the solution incrementally, usually using a specific rule combined with a schedule generation scheme (SGS) to decide which activity to schedule next, ensuring feasibility (Artigues et al., 2026; Hartmann, 2021). Standard schedule generation scheme (SGS) variants include serial, which schedules one activity per step, and parallel, which advances time and initiates as many activities as possible at each step (Artigues et al., 2026; Hartmann, 2021). Common priority rules encompass criteria based on time and slack, such as the latest start time, the latest finish time, and minimum slack, as well as processing- and structure-based criteria, including the shortest processing time and the number of successors (Karam & Lazarova-Molnar, 2013). These types of approaches are simple and fast, often providing good starting points for future scalability, improving the overall schedule (Hartmann, 2021).

However, these heuristics alone leave room for improvement, which motivates other strategies to appear. One of those methods is local search, a technique that starts with an initial solution and tries to explore better neighbors in order to improve that starting schedule. A common pattern is to modify an activity-list representation (e.g., swap/shift activities) and then decode it back into a feasible schedule via an SGS. In simple hill climbing, only improving moves are accepted, which can lead to premature convergence to local optima. More advanced variants (e.g., simulated annealing or tabu search) allow controlled non-improving moves to promote exploration (Das & Acharyya, 2011; Hartmann, 2021).

In the real world, heuristics are frequently used where scheduling interacts with allocation decisions. Capraco et al. (2013) propose a heuristic approach for shipyard space allocation and scheduling that integrates both the handling of spatial restrictions and planning decisions, which suggests how this domain can be explored to obtain better solutions in a complex environment.

Heuristics aim to address the scalability issues raised by exact methods, but there is still space for improvement, as they are problem-specific and can get trapped in local optima. This is where metaheuristic approaches enter, as they can be applied to diverse scheduling types and often produce near-optimal solutions as they explore a broader search space.

3.3.2.2 Population-based metaheuristics

Population-based metaheuristics maintain and progressively improve a set of candidate solutions, allowing exploration of multiple regions of the search space in parallel. In project scheduling, they are particularly attractive when objectives are multi-criteria, feasibility conditions make the search space highly discontinuous, and good solutions are needed under a fixed time budget rather than proven optimality.

Usually, population-based methods are widely chosen approaches (e.g., Genetic Algorithms (GA), Particle Swarm Optimization (PSO), Differential Evolution (DE), Ant Colony Optimization (ACO)) when discussing metaheuristics. A major practical design choice is the solution representation and the associated decoder that maps particles to a feasible schedule. This applies especially under nonrenewable resource constraints, where mode lists may be hard to generate, and evolutionary operators can create infeasible individuals.

In practice, how well these methods work usually depends more on a few key choices: how the problem is set up and solved, how solutions are kept valid, how local search steps are used to improve them, and when to stop or how much time to spend, which has a significant impact on both how good the solutions are and how results compare between studies.

Table 3.8 summarizes, at a high level, when each population-based family is typically a good fit for scheduling and the main engineering pitfalls that determine performance.

Table 3.8: Population-based families in scheduling.

Family	Typical use	Key considerations
GA	Discrete encodings with a strong decoder.	Decoding/repair; intensification (e.g., local search).
PSO / DE	A stable vector-to-schedule mapping exists (discrete variants).	Discretization/mapping; constraint handling.
ACO	Scheduling can be built incrementally (constructive view).	State design; pheromone updates and diversification.

Liu et al. (2025) propose a representative example, the random-key GA, where a feasible mode assignment strategy is used to generate a good initial population, followed by a hierarchical GA framework to balance global and local search. This approach evaluates its performance using a known benchmark library, PSPLIB (Kolisch & Sprecher, 1997), which is essential for the evaluation of several approaches for comparison. The authors also highlight the importance of strong feasibility mechanisms and improvement operators aware of the domain. They report results on multi-mode benchmark classes (e.g., MM50/MM100) and explicitly state instance set sizes (540/540/3240), and they also report new best-known solutions under their experimental settings (Liu et al., 2025), reinforcing that effectiveness hinges on decoding and domain generalization.

Not only are Genetic Algorithms used in an industrial context, but also other swarm and evolutionary strategies perform well in this type of environment. Ramin et al. (2020) propose a unified PSO method for multi-hoist scheduling in galvanic plants to achieve good solutions. This is an approach that is inspired by the social behavior of bird flocking and fish schooling presented by Eberhart and Kennedy (1995).

Also, Chen et al. (2017) use Differential Evolution (DE), an evolutionary algorithm that iteratively improves a population of solutions through operators (e.g., mutation, crossover, selection), combined with an entropy component for RCPSP, showing how these frameworks can improve robustness in the Multi-mode Resource-Constrained Project Scheduling Problem (MRCPSp).

Both these works shown before have something in common: they propose population-based methods in industrial scenarios, combined with specialized approaches that are domain-specific for their problems (Chen et al., 2017; Ramin et al., 2020).

However, as there are multiple metaheuristics for solving the RCPSP, it is important to compare them in similar cases. Van Peteghem and Vanhoucke (2014) provide a comparison of metaheuristics for the RCPSP and introduce a benchmark dataset that supports reproducible evaluation and a stronger comparability across papers. The authors emphasize that performance is influenced by instance characteristics and termination criteria, which underscores the need for standardized benchmarks and transparent reporting.

In the same vein, Das and Acharyya (2011) also compare multiple approaches, highlighting the importance of justifying algorithm choices, rather than relying on generic expectations. As observed, metaheuristics are often combined with other techniques in order to try to get a good

solution. The following section explores that topic, where hybrid approaches are adopted on scheduling problems.

3.3.2.3 Hybrid and matheuristic approaches

As project scheduling models involve multiple objectives, complex operational rules, limited resources, and frequent replanning, single-method heuristics or metaheuristics regularly struggle to secure both strong feasibility and high-quality solutions within practical time constraints. This challenge has led to the development of hybrid approaches that integrate corresponding components for diversification and intensification (Artigues et al., 2026).

In order to achieve both scalability and feasibility, and the best possible solution under complex constraints, other methods emerged. As a result, hybrid and matheuristic approaches (e.g., large neighborhood search (LNS), decomposition, subroutines) combine complementary techniques to better exploit the problem, while keeping computational effort reasonable (Artigues et al., 2026; Hartmann, 2021).

Hybrid algorithms are the main topic, as they combine two or more techniques to solve the problem. Lower in the hierarchy is another type of hybridization, matheuristics, which combines exact methods with metaheuristics (Maniezzo et al., 2021). More precisely, matheuristics integrate mathematical programming or other exact reasoning, such as MILP, CP, or Satisfiability Modulo Theories (SMT) subroutines, into a heuristic/metaheuristic framework, and a key design choice is where exact reasoning is injected (repair, neighborhood evaluation, decomposition, or relaxation) under a given time budget (Maniezzo et al., 2021).

One famous method is the large neighborhood search, a matheuristic used for larger instances, which, instead of making only small local moves, iteratively clears and repairs crucial blocks of a solution. This strategy allows exploration of the solution space while still ensuring feasibility. Borreguero Sanchidrián et al. (2024) use this method in an aeronautical assembly line context, reporting resource-leveling objective reductions above 20% compared with an industrial heuristic baseline, and additional gains of roughly 5-17% over a CP optimizer under time budgets of less than a minute.

Hu et al. (2025) use a decomposition hybrid method, an approach to manage complexity by breaking the problem into smaller, easier-to-manage sub-problems, in order to optimize cost in project scheduling. They propose a constraint programming relax-and-solve algorithm, using decomposition to improve flexibility. For large instances with 280 to 1400 sub-activities, they found a clear trade-off. The proposed CP relax-and-solve approach yields solutions about 0.77% worse than CP, but requires only about 0.15 times the CP runtime. This approach leverages the advantages of both, as it hybridizes an exact model with other strategies, which is common to handle rich constraints in large projects (Hartmann, 2021; Hu et al., 2025).

Lastly, Sánchez et al. (2019) present a Satisfiability Modulo Theories (SMT) solver local search mechanism within an ant colony optimization algorithm, showing a method where the solver pursues rigorous reasoning while the metaheuristic ensures global exploration. They report

improved outcomes on 9 out of 12 benchmark instances (matching in 2 and losing in 1), indicating that solver-guided moves can systematically improve robustness and solution quality.

Hybrids and matheuristics are most effective when feasibility is challenging, time is limited, and plans require frequent updates. In these situations, three common strategies are: destroy-and-repair LNS to escape local minima while maintaining feasibility; decomposition or relax-and-solve to manage scale; and solver-in-the-loop neighborhoods to enforce complex rules during search. These approaches are well-suited to industrial replanning.

3.3.3 Multi-Criteria and Dynamic Scheduling

Many formulations prioritize makespan or a single objective. However, the following sections account for projects from the industrial world, where there are several objectives to be fulfilled. In these cases, planners typically trade off multiple criteria according to their preference, as objectives can be conflicting. Also, maintenance scheduling has to worry about uncertainty, where activities may take longer than predicted, and resources change dynamically, which brings learning-based methods for these complex environments (N. Ma et al., 2023; Ning et al., 2018; Zhang et al., 2023).

Within industrial maintenance planning, three primary design choices guide the selection of optimization approaches: the representation of preferences across objectives, the management of uncertainty, and the extent of plan re-optimization following a disturbance.

3.3.3.1 Multi-objective optimization

When addressing scheduling with multiple objectives, it becomes important to redefine how solutions are evaluated. Instead of a single scalar objective, an objective vector is used. This sets the stage for considering two main approaches: scalarization, which aggregates objectives into a single measure often using weighted sums, and Pareto approaches, which treat objectives separately and seek a set of non-dominated trade-off solutions. Although weighted sums are widespread, they require predetermined weights that may not reflect planners' true preferences. In contrast, Pareto methods generate several reference solutions, allowing preference articulation at decision time (Ning et al., 2018).

Scalarization works best when a feasible plan must be created quickly and improved incrementally, for example by changing penalties or weights. Pareto approaches are better when preferences are unclear or flexible, and a concise, understandable set of options is preferred over listing every possible solution.

San Cristóbal Mateo (2019) presents its approach by building an exact model that accounts not only for time and cost, but also ensures minimum quality and safety risks. Not only does this paper show that multi-objective methods are useful elements for the success of the project, but it also discloses trade-offs between time/cost and safety measures. Operationally, its model quantifies the trade-off between due dates and meeting safety thresholds.

Ning et al. (2018) propose a tri-objective Ant Colony Optimization model for layout planning on a construction site, where they motivate Pareto optimization over weighted sums, to better explore the trade-offs and provide a clearer view for the planner to decide the best option. Also, safety measures are taken into account as an objective to improve.

One specific algorithm is also widely adopted on multi-objective optimization, which is the non-dominated sorting genetic algorithm. This method, used by Ghoddousi et al. (2013), generates Pareto sets for several trade-offs, with the purpose of illustrating how this broad search returns a diverse set of solution schedules, instead of a single one.

3.3.3.2 Dynamic scheduling and replanning

Although having a strong baseline plan is essential, real projects face uncertainty (e.g., failures, variable processing times, resource availability changes). Zhang et al. (2023) show that a problem can be separated into two, according to uncertainty: a baseline scheduling in a deterministic environment, and rescheduling the solution once disturbances occur in a stochastic environment. They adopt that view in carrier-based aircraft maintenance, where their baseline occurs when tasks, modes, resources, and times are fixed and do not account for failures, but once uncertainties enter the problem, rescheduling aims to tackle it (Zhang et al., 2023).

Their reactive approach starts by building a feasible baseline schedule using an improved Non-dominated Sorting Genetic Algorithm II (NSGA-II) variant with local neighborhood search to generate the baseline schemes, and only then applying reactive corrections when malfunctions occur. They propose complete rescheduling and partial rescheduling as responsive mechanisms for the approach, arguing that planners should choose between them depending on the form and scale of uncertainty, in order to explicitly trade off between solution optimality and responsiveness of the method (Zhang et al., 2023).

In industrial replanning, it is important to limit disruption to the plan that has already been communicated, including start times, assignments, and resource commitments. For this reason, reactive rescheduling often uses stability terms that penalize changes from the original plan, balancing the best possible outcome with the need for smooth operations.

In addition to choosing between complete and partial rescheduling, rolling-horizon methods manage the balance between quick responses and stability by locking in the near-term part of the plan and re-optimizing a moving window within time limits. This approach works well when replanning needs to happen often.

Soltan and Ashrafi (2025) also present an approach for the extended Resource-Constrained Multi-Project Scheduling Problem (RCMPSP) by integrating dynamic resource allocation, where instead of having predetermined and fixed resource assignments over time, these can be reassigned based on the project conditions and performance. Their model measures a performance index for the allocation of resources at each period, allowing the number of resources to be different across time. This allows for the solution to be more adaptive than standard approaches that keep quantities constant throughout the problem. They further state that improvements not only depend on the

resource quantity, but also on the prioritization and allocation across projects, which becomes essential to aim for a useful solution.

3.3.3.3 Learning-based techniques

Recent studies explore learning-based scheduling as a promising approach for constantly changing, complex systems. N. Ma et al. (2023) propose a Proximal Policy Optimization (PPO) approach, a popular reinforcement learning algorithm, which is used for refinery crude oil scheduling, highlighting that scheduling has a dynamic decision process. Their PPO agent learns a policy to maximize long term reward, and is then compared with other standard approaches under dynamic environments.

While the above method focused on replacing the optimizer, learning can also be used to adapt the algorithm parameters. Wen et al. (2024) propose a Q-learning-improved genetic algorithm that uses Reinforcement Learning (RL) to adjust operation probabilities (e.g., crossover and mutation), in order to avoid premature convergence and improve its behavior.

Learning-based methods are typically employed in scenarios where decisions are made repeatedly within dynamic environments and a simulator or historical replay is available for training and validation. In the absence of such resources, learning is more effectively utilized as an auxiliary component, for example in parameter control or heuristic selection, rather than as a complete replacement for the core optimizer.

In conclusion, modern scheduling evolved from the focus on one single optimization metric to a multi-objective approach, which helps planners balance the exploration of trade-offs instead of relying on a single target. This strategy is further strengthened by dynamic scheduling, where static plans are constantly changed partially or totally, according to real-world changes to the problem. Lastly, learning-based techniques have been explored as mechanisms to improve adaptivity in scheduling, either by learning scheduling policies or by tuning algorithm behavior.

3.3.4 Comparison of Methods

To support method selection, Table 3.9 summarizes the main families discussed in this chapter and compares them along several criteria.

Table 3.9: Decision matrix for selecting optimization approaches in constrained scheduling (synthesized from the families reviewed in Sections 3.3.1–3.3.3).

Family / Method	Scale	Constraint pressiveness	ex- Time	Guarantees
MILP / ILP	small- medium	medium	minutes-hours	optimal (bounds)
CP	small- medium	high	seconds-hours	optimal (when solved)
Priority-rule heuristics	large	low-medium	milliseconds- seconds	none
Local search	medium- large	low-medium	seconds-minutes	none
Population-based meta- heuristics	medium- large	medium	minutes	none
Hybrid / matheuristics	large	high	seconds-minutes	none (quality under time budget)

Exact approaches yield optimal results on small/medium instances. However, computation time increases rapidly as the number of constraints and their complexity increase. As size grows, algorithms evolve to trade quality for time. Constructive heuristics provide fast initial schedules, while local search and other metaheuristics broaden exploration. Thus, it is essential to focus on the representation to get the best out of the algorithm. Naturally, this leads to hybrid approaches that mix global exploration with local optimality through efficient methods. A typical industrial strategy comprises two-phase solving: an initial baseline construction that addresses hard constraints and stability, followed by an improvement phase using methods such as Local Neighborhood Search or decomposition. This process may also incorporate soft penalties or restricted multi-objective variants to produce a concise set of reference alternatives.

Afterward, it was discussed that scheduling in the real world rarely focuses on a single criterion. Multi-objective formulations are becoming increasingly common, with different approaches to analyze trade-offs among solutions that prioritize different objectives. Lastly, learning-based approaches have been shown to be promising in highly dynamic settings when a simulator exists; otherwise, best used for parameter control.

For high constraint density, constraint programming is a strong default: it expresses rich temporal and resource rules directly and returns a proof of optimality when the search completes within the time budget. This constraint-programming core can be combined with time-bounded improvement for larger instances, for example large-neighborhood search or decomposition, whether applied as separate phases or as strategies running inside the solver itself. When the scope can be bounded to small or medium subproblems with stable data and a strong need for cost-driven trade-offs and certificates, MILP and CP remain suitable as exact cores or subsolvers.

3.4 Summary

The literature reviewed in this chapter is mature on each side of the problem but rarely joins the two. On the visualization side, it points to a shift from merely observing schedules to understanding them at scale: explaining feasibility through dependencies and managing change through iterative refinement, with the most effective solutions pairing expressive representations with interaction strategies such as aggregation, coordinated views, and what-if comparison (Section 3.2). On the optimization side, exact methods such as constraint and mixed-integer programming offer optimality certificates on small-to-medium instances but trade quality for time as size grows, where heuristics, metaheuristics, and hybrid two-phase schemes take over; for the high constraint density of a programmed stop, constraint programming stands out, expressing rich temporal and resource rules directly and returning a proof of optimality when the search completes within budget (Section 3.3).

The gap is the integration. Visualization tools rarely optimize, optimization tools rarely offer the visual, interactive reasoning that planners rely on, and neither builds on a reusable abstraction of a plan that recurs from one stop to the next. Table 3.10 makes this concrete, setting the representative systems reviewed above against the four capabilities that a solution to the CERN problem must combine: no existing tool spans all four, and the reusable-template column is empty except for this work.

Table 3.10: Representative planning, visualization, and scheduling systems against the four capabilities this work integrates. ✓ supported, – not offered or not reported in the source.

System	Reusable template	Schedule optimization	Interactive review	Toolchain integration
Microsoft Project	–	✓	✓	✓
Chronos	–	–	✓	–
LiveGantt	–	✓	✓	–
CasCADE	–	–	✓	✓
PlanningVis	–	✓	✓	–
A-Plan	–	✓	✓	–
Cobi	–	✓	✓	✓
This work	✓	✓	✓	✓

This gap is what the two research questions of this dissertation address, one on the visual representation of constraint-rich schedules and one on their modeling and optimization, and it is what the next chapter sets out to close, by realizing a constraint-programming core and a constraint-aware interactive interface over a reusable template for the CERN planning problem.

Chapter 4

Solution Specification

The preceding chapters established the planning problem at CERN and surveyed the visualization and optimization techniques relevant to it. The decision matrix that closed the optimization survey (Table 3.9) identified constraint programming as the family that combines the highest constraint expressiveness with optimality guarantees when a solution is proven, at the cost of scaling only to small and medium instances. The visualization survey concluded, in turn, that timeline representations are a sound starting point but are insufficient on their own, that coordinated perspectives and semantically transparent notations are required to expose dependencies and conflicts, and that interactivity is decisive when coordinators must compare alternatives and reach a decision (Moody, 2009; Sakin & Isaacs, 2024; Sun et al., 2020). This chapter specifies the solution that was built on those foundations. It opens with the problem it addresses, its constraints, and the requirements it must meet, then defines the template abstraction at its conceptual core, the architecture around it, and the design of its three components, foregrounding the decisions behind each. The concrete realization of this specification — the data model, the solver encoding, the engineering of the interface, and the technologies — is described in Chapter 5.

The solution is realized as a web application that extends CERN’s existing Scheduling Tools stack rather than as a standalone application, so that it consumes the same data and fits the workflow that coordinators already use. It is organized around three components, a reusable *template engine*, a constraint-programming *optimizer*, and an interactive *review and export interface*, the same three that the evaluation protocol sets out to assess (Chapter 6). The template abstraction is the conceptual core of the work: it is the artifact that coordinators author and reuse, it is the structural input on which the optimizer operates at solve time, it is what the interface renders, and it is what other CERN systems can instantiate through its named parameters (Section 4.3.2). The optimizer and the interface are therefore best understood as the two engines that act upon templates: one computational, which proposes optimized schedules, and one human, through which the coordinator inspects, corrects, and commits them.

4.1 Approach Overview

The approach supports two complementary flows that mirror the way coordinators work. In the first flow, a coordinator *captures* a recurring group of activities once, as a template: a named, hierarchical set of tasks carrying attributes such as their durations, dependencies, responsible teams, locations, and scheduling constraints. In the second flow, a coordinator *composes* a new schedule by instantiating one or more templates onto a calendar, adjusts it to the case at hand by setting resource limits, by adding dependencies within and between instances, and by editing individual tasks where needed, optimizes the result, reviews the proposed schedule task by task, and exports the result to Microsoft Project, the interchange format of CERN's existing planning tools. Because the Scheduling Tools database from which the templates are captured is itself refreshed daily from Microsoft Project, the two flows close a loop within the existing toolchain rather than standing apart from it. Figure 4.1 summarizes the two flows.

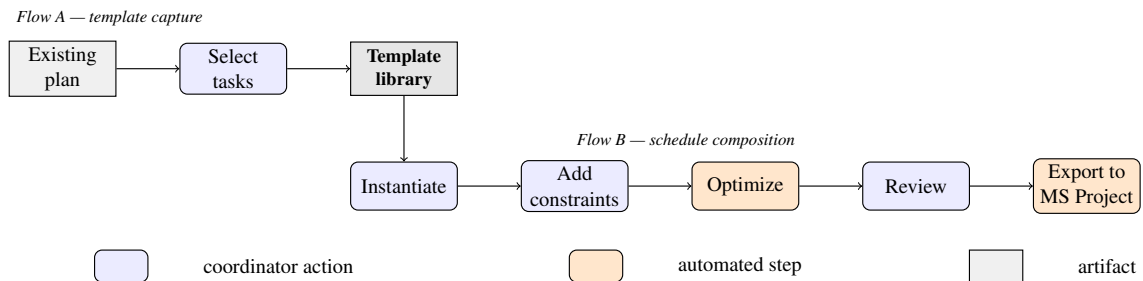


Figure 4.1: The two application flows: template capture (A) and schedule composition (B). Rounded blue boxes are actions the coordinator performs, rounded orange boxes are automated steps, and gray rectangles are artifacts.

This design gives concrete form to the central entities of the planning vocabulary introduced in the background chapter (Table 2.4), modeling those that the scheduling problem at hand requires. A *task* becomes a row in a template or in a schedule instance, carrying a duration, a hierarchy, and a scheduling constraint. A *dependency* becomes a typed, lagged edge between two tasks. *Resources* are realized as teams and locations with concurrency limits. The *calendar* is realized as a shared working-time model with CERN holidays. The *objective* is realized as the quantity that the optimizer minimizes. The remainder of this chapter specifies how each of these is represented, reasoned about, and presented.

4.2 Problem and Requirements

Before the approach is specified, the problem it addresses must be stated precisely and the requirements that follow from it made explicit. This section characterizes the planning of a programmed stop, delimits which of its constraints fall within the scope of this work and which lie outside it, and distills from them the requirements that the solution must meet.

4.2.1 The Planning Problem

Planning a programmed stop means turning a large set of maintenance interventions into a dated schedule that respects how the work must actually unfold. A year-end stop of a single accelerator already involves several hundred leaf tasks, organized in a hierarchy of summary tasks and their sub-tasks, and the count grows with the scope of the stop and the number of machines it touches. Each task carries a duration, a responsible team, and, frequently, a physical location. The tasks are not independent: they are bound into a network by precedence relations, and many compete for the same teams and the same locations, while all of them advance only during working time, around weekends, public holidays, and the shift patterns of the groups that carry them out. Today this scheduling is performed largely by hand in Microsoft Project, a tool that records a schedule but offers little help in reasoning about how the constraints interact; and because the broad structure of a stop recurs from one year to the next, much of the work is reconstructed from scratch each time. The coordinator's task is to find an ordering and a set of dates that satisfy every constraint while keeping the machines off for as little time as possible.

4.2.2 Scope of the Planning Problem

The constraints that shape such a plan fall into a few kinds, which together fix the feasibility and the length of a schedule; a re-implementable statement of the problem requires each to be made explicit.

- **Precedence.** Tasks are ordered by dependencies in the four patterns of the precedence diagramming method — finish-to-start, start-to-start, finish-to-finish, and start-to-finish — each optionally carrying a positive or negative lag, so that one task cannot start, or finish, until a fixed amount of time after another has finished, or started.
- **Working-time calendar.** Most tasks consume *working time*: they advance only during working hours, around the lunch break, the night, weekends, and CERN holidays. Others *elapse* continuously, every hour of every day, because the physical process they represent — a cooldown, the decay of induced radioactivity — does not observe working hours. The two kinds are linked by dependencies and by the makespan, so a schedule must reason about both on a common scale.
- **Date constraints.** A task may be anchored to the calendar independently of its predecessors: required to start or finish no earlier than, no later than, or exactly on a given date. Such anchors express external commitments — a beam stop fixed for a particular date, a delivery that cannot arrive sooner — that the ordering alone does not capture.
- **Resource concurrency.** Teams and physical locations have limited capacity: only so many crews of a given team can work in parallel, and only so many activities can share a location at once, which caps how much work may proceed simultaneously.

- **Locks.** The coordinator may pin a task, or a whole sub-project, to fixed dates, removing it from the optimizer’s discretion — a means of asserting operational knowledge the model does not hold.

This work addresses these constraints, which together account for the temporal-resource envelope of a stop. A further set governs CERN interventions but lies outside the model: the safety, access, and radiation-protection requirements — permits, training prerequisites, dose limits — administered by dedicated CERN processes and tools such as IMPACT (Section 2.2.2), and the spatial interference between nearby interventions, currently checked by separate spatial tooling. Drawing the boundary here is a scope decision rather than a claim that those dimensions are solved elsewhere: a conflict detected by those tools still has to be resolved by re-scheduling, so folding them into the model is a direction the conclusions return to (Chapter 8).

4.2.3 Requirements

From this problem, the approach must meet four requirements, around which the rest of the chapter and the evaluation are organized:

- **R1, reuse.** Capture a recurring plan once, as an editable abstraction, and instantiate it onto a new stop without rebuilding it by hand.
- **R2, optimization.** Compute, from such an instance and its constraints, a schedule that minimizes the makespan over working and calendar time, with an optimality guarantee where one is attainable.
- **R3, review.** Present the optimized schedule, its constraints, and the changes it proposes so that the coordinator can inspect, correct, and commit them, keeping the human in control of the result.
- **R4, integration.** Consume and produce the Microsoft Project files that the existing CERN toolchain already exchanges, so that the result fits the established workflow rather than replacing it.

These four requirements map onto the three components introduced above: the template engine answers R1, the optimizer answers R2, and the review and export interface answers R3 and R4. They are also what Chapters 6 and 7 set out to evaluate.

Table 4.1 refines these four themes into the concrete functional and non-functional requirements that the design must satisfy.

4.3 The Template Abstraction

The central premise of this work is that a large fraction of the activities planned for a technical stop is recurrent. A given intervention decomposes into much the same ordered set of tasks each

Table 4.1: Functional and non-functional requirements, refining the four high-level requirements R1–R4.

ID	Theme	Requirement
<i>Functional</i>		
FR1	Reuse (R1)	Capture a recurring plan as a reusable, parameterized template and instantiate it onto a stop, individually or in batch.
FR2	Reuse (R1)	Support individual, one-off tasks alongside templated work, so not every task must belong to a template.
FR3	Optimization (R2)	Express the constraint set: precedence (FS/SS/FF/SF with lags), date constraints, locks, and resource concurrency limits.
FR4	Optimization (R2)	Compute a makespan-minimizing schedule with an optimality guarantee where attainable, and report infeasibility as fixable violations.
FR5	Review (R3)	Present the proposed schedule and its changes for task-by-task review, acceptance, or reversion.
FR6	Integration (R4)	Consume plans originating in Microsoft Project (via the platform database) and export Microsoft Project files.
<i>Non-functional</i>		
NFR1	Performance	Solve a single-stop instance (hundreds of tasks) within a practical time budget.
NFR2	Integration	Run as an extension of the existing Scheduling Tools platform, reusing its client, backend, and database.
NFR3	Usability	Be operable by coordinators after a short introduction, independent of planning background.
NFR4	Interpretability	Make solver outcomes, including infeasibility, explainable to the coordinator.

time it is planned: the same teams, in the same order, with roughly the same durations. The recurrence runs in two directions: the same intervention returns from one stop to the next, and, within a single stop, the same procedure is applied to many pieces of equipment of the same kind, so one template is typically instantiated several times in the same schedule, once per magnet, per sector, or per system (Figure 4.2). Chapter 7 quantifies this recurrence on a real stop. The template abstraction captures it once so that it can be reused, validated, and reasoned about, instead of being rebuilt by hand for every new schedule. This premise is one of degree, not of exclusivity. A large fraction of the work is recurrent, but not all of it, and the abstraction is not a straitjacket: not every task need belong to a template. The workflow accommodates individual, one-off tasks as well, and a real stop typically mixes recurring, templated work with stop-specific activity that no template covers, as the case study shows (Section 7.5).

Reuse here is more than the static, file-level copying that conventional planning tools support, where a saved plan is duplicated and then edited by hand for each new case. A template is a first-class, parameterized entity: it is simultaneously the object a coordinator authors, the structured input the optimizer schedules (Section 4.5), the entity the interactive timeline renders (Section 4.6), and, through its named and typed placeholders (Section 4.3.2), the contract through which other CERN systems instantiate it programmatically. Reuse is therefore not a convenience layered onto a planning tool but the single abstraction that ties the entire workflow together.

This section sets out the design of the template abstraction: how it is represented, how it is

parameterized for reuse, and the design decisions behind both. Its realization — how templates are stored, captured, instantiated, and governed — is the subject of the template engine in Chapter 5.

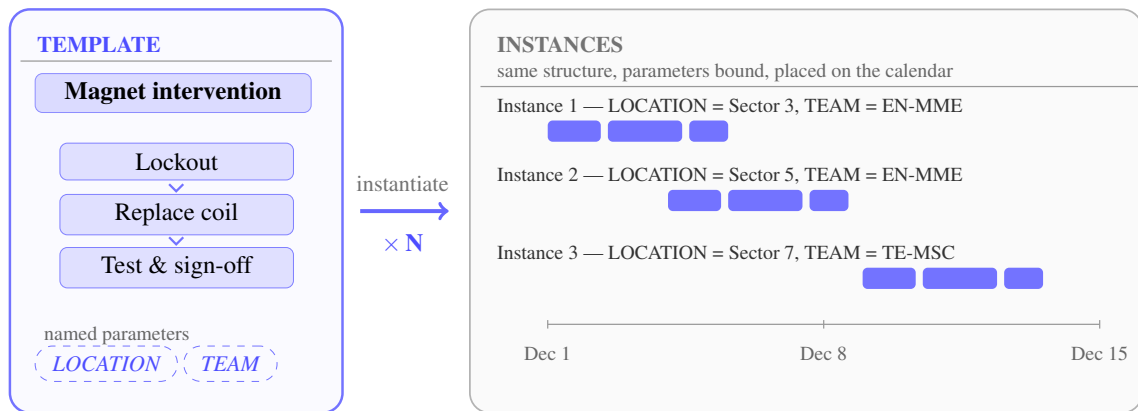


Figure 4.2: The template as a reusable, parameterized blueprint. Authored once, a template carries a hierarchy of tasks and a set of named parameters; it is then instantiated any number of times, each instance binding the parameters — here a location and a team — to concrete values and placed on the calendar. The same template is reused both within a single stop, once per magnet, sector, or system, and across successive stops.

4.3.1 Absolute Dates versus Relative Offsets

A template stores, for every task, the *absolute* start and finish as they were captured, anchored to a base date, rather than offsets relative to that base. The alternative — normalizing a template to relative offsets at capture time, so that a task would be recorded as starting, for example, two working days after the template origin, and materializing concrete dates only in the instances — was considered and rejected, for reasons that follow from how templates are produced and consumed. A template is captured from a real plan, so absolute dates are what the source data contains; normalizing at capture would discard the calendar context against which the captured layout was valid and would commit, at that moment, to one canonical origin. Keeping the captured dates makes the capture lossless and defers the only conversion actually needed, the temporal shift, to instantiation time, when the true anchor, the requested start date, is known. Three further properties follow. The scheduling constraints of Microsoft Project carry absolute dates, and those dates survive the round trip through the template unchanged. The distinction between tasks that consume working time and tasks that elapse continuously (Section 4.5.1) is only well defined against a real calendar, and the shift arithmetic needs concrete dates on both sides. And a template stored in the same representation as a schedule shares the schedule’s validation and rendering paths, instead of requiring a second, offset-based code path for every rule. The decision also has costs, which are accepted consciously: the stored dates are historical, carrying no meaning for the next stop until the template is re-anchored, and a direct reader of the stored template sees a fragment of an old calendar rather than a neutral pattern. The editor compensates by presenting a template as a self-contained layout, with dates shown relative to its own start; the stored calendar

position is invisible to the author. The remaining cost is the shift computation at instantiation, whose implementation is described in Section 5.2.3.

Separating templates from schedules. Templates and schedules are kept as separate entities. The separation reflects two genuinely different roles. A template is a shared asset, reused across many schedules and edited by the managers of the facility that owns it; a schedule is a working object that belongs to one particular plan. The two are independent: editing an instance within a schedule changes only that schedule and never propagates back to the template from which the instance was created. Collapsing both into one entity with a discriminator field was considered and rejected, because it would have entangled the permissions and the editing history of the two, for no structural gain.

4.3.2 Parameterized Templates

The reuse a template offers is broader than repeating an identical plan. The procedure to service one magnet is the procedure to service the next, except that the magnet, the responsible team, or an equipment code changes; what varies is not a digit but a small set of identifiable values. To capture this, a template may declare *placeholders*: named, typed points of variability that are left unbound in the template and receive concrete values when it is instantiated. The procedure is thereby authored once, and binding its placeholders adapts it to any magnet, any team, or any equipment code. Each placeholder has a name and a type — location, group (team), or free text — and the semantics are deliberately those of a logic variable: every occurrence of a given placeholder receives the same binding, exactly as every occurrence of a variable in a logic clause is unified to the same term.

Three design decisions shape this model. First, references are recorded *per occurrence* rather than globally: a real template may mention the same equipment identifier in many tasks while only some of those tasks are genuinely variable, so a global “replace every occurrence” rule would be incorrect, and the author marks each variability point explicitly. Second, a reference may be either a *replacement* of an existing value or an *injection* into a task that has no such value, which lets a task declare a field as parametric even when the template carries no default for it. Third, the model is stored relationally rather than as an opaque document, which keeps the historical bindings queryable.

Placeholders as an integration point. The value of an explicit, named parameter model extends beyond the editor. Once a template exposes its variability as a list of named, typed parameters, the act of instantiating it becomes a well-defined operation whose inputs are exactly those parameters. Other CERN systems can then enumerate the templates available for a facility and request an instantiation by supplying a value for each parameter, attaching the result to a specific piece of equipment. The placeholders are, in this sense, the contract through which the template engine is consumed programmatically, and the relational binding records are what allow a downstream module to ask which activities are planned for a given piece of equipment.

Governing template trustworthiness. Because templates are shared assets that feed every schedule built from them, their trustworthiness must be governed: a coordinator composing a schedule needs to know whether a template has been validated, by whom, and when, so as not to build on a draft that is still being edited. The approach therefore models a publication lifecycle for templates, through the states draft, in review, approved, and archived. The realization of the model, the capture, the instantiation, the parameterization, and the lifecycle just described is the subject of Section 5.2.

4.4 Conceptual Architecture

The architecture comprises four cooperating parts, shown in Fig. 4.3: a single-page web client, a backend application, a relational database, and a separate optimization service. The web client renders the templates, the schedules, and the interactive timeline. The backend application exposes a Representational State Transfer (REST) application programming interface (API), owns the domain model and its persistence, and orchestrates optimization. The database stores templates, schedules, and the full record of the entire optimization. The optimization service is a self-contained microservice that receives a schedule, solves it, and returns a proposed date assignment. Of these four, the web client, the backend, and the database *extend* the existing Scheduling Tools platform, whereas the optimization service and the Microsoft Project export are *new* to this work (Fig. 4.3).

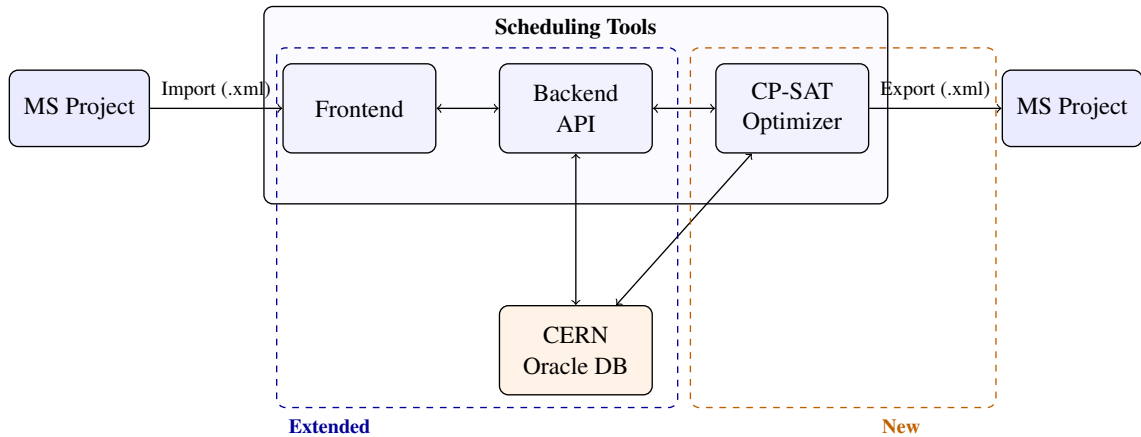


Figure 4.3: High-level architecture of the prototype.

Separation of the optimizer. The most consequential architectural decision was to place the optimizer in a separate service, written in Python, rather than inside the backend application. The decision was driven by the solver technology. The model relies on constructs that are most completely and most expressively supported by the Python binding of Google OR-Tools, namely interval variables, the `AddElement` table constraint, and the cumulative and no-overlap global constraints (Perron & Didier, 2025). Embedding the solver in the backend would have required

either a partial native binding or in-process execution, the latter of which would have coupled the lifetime of a long solve to the lifetime of an application thread and would have made cancellation difficult. Exposing the solver as a Hypertext Transfer Protocol (HTTP) microservice instead yields three properties that matter operationally: failure isolation, since a solver that exhausts memory or crashes does not take the application down; controllable cancellation, since the service can be signaled to stop a running solve and return the best schedule found so far; and independent deployment and resource quotas, since the solver can be scaled and limited separately from the rest of the stack. The cost of this decision is the serialization of a potentially large request and the management of a long-lived request window, both of which the implementation addresses (Section 5.1); this transport cost is independent of the solver’s own scaling, which is characterized in the algorithmic evaluation (Chapter 7).

Technology choices. The remaining decisions concern the technologies. The web client, the backend, and the database are inherited from the Scheduling Tools platform that the approach extends; adopting them was a condition of the integration rather than a free choice, and it is what keeps the result deployable alongside the existing tool. CP-SAT was selected among constraint solvers because it is open source and actively developed and because the interval, table, and cumulative constraints on which the scheduling formulation of Section 4.5 rests map directly onto its interface, where the main alternative of comparable modeling power, IBM ILOG CP Optimizer, is a commercial product. The Microsoft Project export is generated with MPXJ, the only established open-source library for the format, which spares the export from hand-writing the broad and error-prone Extensible Markup Language (XML) interchange schema. The concrete technology stack and the deployment are detailed in Chapter 5.

4.5 The Scheduling Problem

The optimizer assigns concrete dates to the tasks of a schedule so that every hard constraint is satisfied and the schedule is as short as the constraints allow. In the terms of the optimization survey, the problem is a resource-constrained project scheduling problem enriched with the generalized precedence relations and the calendar constraints that the CERN setting requires (Artigues et al., 2026; Hartmann, 2021); in the three-field notation of Section 2.1.1, it corresponds to $PS|temp,cal|C_{max}$, project scheduling with generalized precedence relations and calendars under makespan minimization, the class for which specialized calendar procedures have been developed (Kreter et al., 2015). Among the families surveyed, constraint programming was selected. The choice deliberately weighs expressiveness above raw scalability, and four observations support it. First, the model is dominated by rich logical constraints rather than by a numeric cost structure: the working-time calendar, the four generalized precedence relations, the date constraints, the locks, and the concurrency limits can all be stated directly as constraints, whereas the metaheuristic families that scale furthest would require a dedicated encoding and repair scheme for each of them, and, on their own, certify neither infeasibility nor how far a solution lies from the

optimum (Hu et al., 2025; Karam & Lazarova-Molnar, 2013). Second, the instances that matter operationally are of small and medium size: a year-end technical stop, the kind of stop in which recurring work concentrates and where templates therefore apply best, comprises hundreds of leaf tasks rather than tens of thousands, which is precisely the regime in which constraint programming is strongest (Table 3.9). Third, the classical scaling limitation of the family is tempered in CP-SAT itself (constraint programming over Boolean satisfiability, the solver of Google OR-Tools), whose portfolio runs exact constraint propagation alongside large-neighborhood-search and linear-relaxation workers, combining in one solver the exact core and the improvement heuristics that the survey associates with larger instances. Fourth, the metaheuristic and matheuristic families that scale furthest are, in the surveyed literature, associated with a different operating regime: very large instances, or settings in which a good solution must be produced under a tight time budget and the problem is re-solved frequently (Maniezzo et al., 2021). A programmed technical stop is planned offline, where solve time is not the binding constraint, so an exact method is preferable to a fast approximate repair. Moreover, some of the hybrid reasoning that benefits larger instances is already available within CP-SAT, as noted above, so selecting constraint programming does not preclude it. A proof of optimality, or a bounded gap, is also of operational value in itself: it tells the coordinator that no shorter schedule exists under the declared constraints. The scale at which the resulting model remains practical, and the boundary of the approach, are characterized in the algorithmic evaluation (Chapter 7). While the individual constraint forms are standard, two aspects of the model demand more than a routine encoding and are where the technical effort of this dissertation concentrates: reconciling working and continuously-elapsing time on a single scale, and turning an infeasible model from an opaque rejection into an intelligible, actionable explanation. The subsections that follow specify the task model, the constraints, and the objective, together with the requirement that infeasibility be explained; the encoding of this problem in the solver, and the search configuration, are described in Chapter 5.

4.5.1 Tasks and Time

Two kinds of task coexist in a CERN schedule and must be scheduled together. A *regular* task consumes working time: it progresses only during the eight working hours of a weekday and is suspended over the lunch break, the night, the weekend, and the holidays. In the CERN configuration, working time runs from 08:00 to 12:00 and from 13:00 to 17:00, Monday to Friday, excluding CERN holidays; the calendar is a configurable parameter of the approach rather than a fixed assumption. An *elapsed* task progresses continuously, every hour of every day, because the physical process it represents, such as a cooldown or the decay of induced radioactivity, does not observe working hours. Dependencies and the overall makespan link the two kinds freely, so any model of the problem must reason about both on a common scale. Reconciling these two notions of time is the central modeling problem the formulation must solve, and the dual working- and calendar-time encoding developed for it is presented in Chapter 5.

4.5.2 Constraints

Section 4.2.2 described these constraints as they arise in the problem; this section states them formally, as they enter the optimizer’s model. The optimizer enforces the dependency relations, the per-task scheduling constraints, the task locks, and the concurrency limits. Table 4.2 lists the scheduling constraint types and how each is expressed in the model. The four dependency types of the precedence diagramming method are supported, each with a lag that may itself be expressed in working or in elapsed units.

Table 4.2: Scheduling constraint types supported by the optimizer and their encoding.

Category	Constraint	Encoding in the model
Dependency	Finish-to-Start (FS)	$\text{successor start} \geq \text{predecessor end} + \text{lag}$
Dependency	Start-to-Start (SS)	$\text{successor start} \geq \text{predecessor start} + \text{lag}$
Dependency	Finish-to-Finish (FF)	$\text{successor end} \geq \text{predecessor end} + \text{lag}$
Dependency	Start-to-Finish (SF)	$\text{successor end} \geq \text{predecessor start} + \text{lag}$
Date	Start No Earlier Than (SNET)	$\text{start} \geq \text{offset}$
Date	Start No Later Than (SNLT)	$\text{start} \leq \text{offset}$
Date	Finish No Earlier Than (FNET)	$\text{end} \geq \text{offset}$
Date	Finish No Later Than (FNLT)	$\text{end} \leq \text{offset}$
Date	Must Start On (MSO)	$\text{start} = \text{offset}$
Date	Must Finish On (MFO)	$\text{end} = \text{offset}$
Preference	As Soon As Possible (ASAP)	soft preference in the objective
Preference	As Late As Possible (ALAP)	soft preference in the objective

Locks. Locks express the coordinator’s control over the solve. A locked task is pinned to its current dates and is treated by the solver as a fixed interval; a container lock, applied to a summary task, forces all of its descendants to remain within its span, which lets a coordinator freeze the envelope of a sub-project while still allowing its internal tasks to be rearranged. Locks differ in kind from the constraints above: a date constraint bounds where a task may go, whereas a lock removes the task from the optimizer’s discretion altogether.

Concurrency limits. Concurrency limits cap how many tasks may run at the same time at a given location or within a given team: a location limit expresses how many activities that physical area can host at once, and a team limit expresses how many crews of that team are available to work in parallel. They bound the schedule’s parallelism on its resource dimension. The model enforces them at a deliberately coarse temporal resolution, trading some precision on these limits alone for tractability; the encoding and the reasoning behind that resolution are described in Section 5.3.3.

The constraints in Table 4.2 cover the temporal and resource dimensions of the problem; the safety, access, radiation-protection, and spatial requirements that lie outside the model are described in Section 4.2.2. The present optimizer targets the temporal-resource core of the problem, on which the feasibility and the length of a schedule principally depend.

4.5.3 Objective

The optimizer minimizes the makespan of the schedule, defined as the latest calendar-minute end over all tasks. Although the window of a programmed stop is largely fixed, minimizing the makespan is a deliberate first objective: a more compact schedule leaves slack within that window to absorb delays, and the minimum makespan answers a question the coordinator needs first: whether the planned work fits within the available window, and with how much margin. Formally, over the set of tasks T the solver minimizes

$$W \cdot \max_{t \in T} \text{end}_t^{\text{cal}} + \sum_{t \in T} w_t \text{start}_t,$$

where $\text{end}_t^{\text{cal}}$ is the calendar end of task t , the first term is the makespan scaled by a large factor W , and the per-task weights w_t realize the two preference constraints of Table 4.2: w_t is positive for a task marked as-soon-as-possible, pulling it earlier; negative for a task marked as-late-as-possible, pushing it later; and a small positive default otherwise, which gently prefers an early start and so avoids leaving tasks needlessly spread out. The factor W is set several orders of magnitude above the per-task weights, large enough that the secondary term never trades against the makespan: an as-late-as-possible preference consumes available slack but never extends the schedule. The objective is structured to admit further criteria, such as resource leveling and the minimization of radiation exposure; the present version optimizes the makespan alone, with the multi-objective extension outside the scope of this work.

4.5.4 Explaining Infeasibility

A solver that answers only “infeasible” is of little use to a coordinator, who is left without any indication of which of the dozens of constraints just declared are in conflict (Fig. 4.4). Replacing that opaque answer with a structured, actionable explanation is therefore both a requirement of the solution and, as the second of the two aspects identified above, a genuine technical challenge rather than a routine encoding step. It is a step towards the interpretability the planning context demands: every infeasibility is reported as a typed violation, paired where possible with a suggested, reversible fix that the coordinator can apply with one click. The diagnostic procedure that delivers this is developed in Chapter 5.

4.6 Visualization Design

The optimizer proposes; the coordinator decides. The interface is where a coordinator reads a schedule, understands what the optimizer changed and why, accepts or overrides those changes, and exports the result. The visualization survey motivated two commitments that shape this interface: that the timeline is the right foundation but must be enriched to carry the logic of dependencies and constraints, and that the symbols used for those constraints should suggest their

```

Problem closed by presolve.
CpSolverResponse summary:
status: INFEASIBLE
objective: NA
best_bound: NA
integers: 0
booleans: 0
conflicts: 0
branches: 0
propagations: 0
integer_propagations: 0
restarts: 0
lp_iterations: 0
walltime: 0.0660643
usertime: 0.0660646
deterministic_time: 0
gap_integral: 0

2026-06-03 22:55:15,951 [INFO] solver.scheduler: Solver finished in
0.07s with status=INFEASIBLE
2026-06-03 22:55:15,951 [ERROR] solver.scheduler: Solver returned
INFEASIBLE (code=3) after 0.07s. Model has 559 variables and 657
constraints.

```

Figure 4.4: The opaque solver response on an infeasible model: a bare INFEASIBLE status with no indication of the cause.

meaning rather than being arbitrary marks the coordinator must memorize, following the principle of semantic transparency (Aigner et al., 2011; Moody, 2009).

Enriched timeline. The interface presents the schedule on a single Gantt timeline, enriched with the visual encodings described below. Concentrating the information on one coherent surface keeps the coordinator’s attention in a single place and avoids the interaction cost of moving between separate views; it also lets the prototype extend the very Gantt component that the Scheduling Tools platform already uses, so that the result is continuous with the tool the coordinators know. The enrichments give the single timeline much of the analytical power that separate dependency, resource, and constraint views would provide, at a fraction of the cognitive overhead.

The visual language comprises several coordinated encodings. The four dependency types are drawn as arrows distinguished by color, each labeled with its lag. The scheduling constraints are drawn in place against the tasks they govern: the soft preferences as pills beside the bar, and the date constraints as glyphs at the constrained date whose shape encodes their meaning. Each date glyph pairs a vertical stroke marking the bound date with two arms: the top arm points toward the task bar and so distinguishes a start constraint from a finish constraint, while the bottom arm points in the direction the bound endpoint may still move, so that the pair no-earlier/no-later is read directly off the glyph. The must-start-on and must-finish-on constraints are drawn as parentheses hugging the pinned endpoint. A locked task is drawn with a hatched fill and a padlock marker above its bar, with distinct markers for a pinned leaf and for a container lock. Concurrency pressure is shown as a heatmap beneath the bars (described below). Weekends and CERN holidays are shaded, so that the working calendar is visible rather than implicit.

Concurrency heatmap. The concurrency heatmap lets a coordinator see at a glance where a location or a team is overcommitted. Each resource that carries a limit is drawn as a row beneath the timeline, showing for every day the peak number of tasks competing for that resource; the row can be focused on a single resource of interest. A row presents three states: a resource over its limit is flagged in red with a warning marker, a resource exactly at its limit is flagged in orange, since one more task would exceed it, and a resource below its limit carries no marker.

Before-and-after overlay. The single most useful encoding for review is the overlay of the schedule before and after optimization. The positions a task held before the solve are drawn as a faint, semi-transparent bar behind the optimized bar, so that the displacement the optimizer introduced is visible directly on the timeline. The displacement is also enumerated in a list of changes, grouped by instance, in which each moved task carries a chip naming the reason the optimizer moved it: a dependency, a scheduling constraint, a concurrency limit, or a lock. Selecting a change focuses the corresponding task on the timeline. This pairing of an at-a-glance overlay with an itemized, reason-annotated list is what turns the optimizer's output from a result the coordinator must simply accept into one they can interrogate.

4.7 The Review Workflow

The interface treats an optimization not as a command to be obeyed but as a proposal to be adjudicated, because a coordinator holds operational knowledge that the model does not encode and must retain the final say. This human-in-the-loop stance is realized as an explicit decision workflow recorded against every optimization.

A proposal has three possible outcomes: it may be accepted, rejected, or, once accepted, later reverted. Accepting writes the optimized dates onto the schedule, rejecting leaves it untouched, and reverting an accepted optimization restores the dates that preceded it. A schedule carries at most one undecided proposal at a time, so that the coordinator is never confronted with competing optimizations, and any edit that would invalidate an undecided proposal asks for confirmation before it proceeds. How a proposal is persisted, so that it can be reverted without a separate snapshot, is described in Section 5.4.1.

Acceptance is not all-or-nothing. In a review mode, the coordinator may accept some of the optimizer's changes and decline others, task by task, and may additionally drag a task to a position of their own choosing, with the manual placement validated against the same working-time arithmetic used everywhere else. As the coordinator toggles changes and drags tasks, the dependency and concurrency violations implied by the current, partly manual schedule are recomputed and shown live, so that the consequences of an override are visible before it is committed. The intent of this workflow is that the approach support a negotiation between the coordinator and the optimizer rather than impose a single opaque result.

The edit mode carries a visual language of its own. A manual placement is marked in the change list with a *Manual* badge, through which it can be individually reverted; a task whose

current position violates a dependency or a concurrency limit is highlighted on the timeline and flagged with a warning marker, and the corresponding entry in the changes panel names the violated relation. Because the violations are recomputed as the coordinator drags, the marking always reflects the schedule as it currently stands, not as it stood when the optimizer returned.

4.8 Summary

This chapter specified the approach as a single, template-centric workflow. Its conceptual core is the template abstraction: a reusable, parameterized plan that serves at once as the coordinator's editable artifact and as the structured input to the optimizer (Section 4.3). Around it, a conceptual architecture separates a constraint-programming optimizer, expressed over a dual working- and calendar-time model, from an interactive interface that renders the schedule, its constraints, and the proposed changes for review, and integrates back to Microsoft Project to keep the result inside the existing toolchain (Sections 4.4–4.7). Together these meet the four requirements set out in Section 4.2: reuse, optimization, review, and integration. The next chapter describes how this specification was realized.

Chapter 5

Implementation

The previous chapter specified the solution: its flows, its architecture, and the design of the template engine, the optimizer, and the review interface, together with the decisions behind them. This chapter describes how that specification is realized — the technology stack, the mechanics of the template engine, the encoding of the scheduling problem in the solver, the engineering of the interactive interface, and the difficulties encountered along the way, which are drawn together in Section 5.5.

5.1 Technology Stack and Deployment

The backend follows the layered, gateway-oriented structure dictated by the existing codebase: controllers translate HTTP requests into calls on services, services hold the domain logic, and gateways abstract every outbound dependency, namely the database, the optimization service, and the Microsoft Project file format. This arrangement keeps the optimizer and the file exporter behind interfaces and makes the domain logic independent of them. Table 5.1 lists the principal technologies of the prototype.

Table 5.1: Principal technologies of the prototype.

Part	Technologies	Origin
Web client	React, TypeScript, the DHTMLX Gantt component (DHTMLX, 2021)	existing platform
Backend	Java, Spring Boot, layered services and gateways	existing platform
Persistence	Oracle relational database	existing platform
Optimizer	Python, Flask, Google OR-Tools CP-SAT (Perron & Didier, 2025)	this work
Interoperability	MPXJ for Microsoft Project file generation (Iles, 2019)	this work

The rationale for these choices, reusing the platform’s client, backend, and database, selecting CP-SAT among constraint solvers, and using MPXJ for the Microsoft Project format, is given with the architectural decisions in Section 4.4. At the level of realization, the optimizer is a thin Flask

wrapper around CP-SAT through its Python binding, the most complete interface to the solver, holding no logic beyond marshalling a schedule into the model and the solution back out.

The optimization service is packaged as a container and runs on CERN's OpenShift infrastructure, where it is reached over the cluster's internal network and is never exposed externally. During the evaluation it ran in a development environment against a test instance of the Scheduling Tools database, with completed and reviewed features promoted to the pre-production cluster incrementally; at the time of writing it sits in pre-production, validated but not yet in day-to-day operational use. Because a solve can run for many seconds over a large request, the backend guards each call to the service with a fast health probe and a retry policy that distinguishes server from client errors; the JavaScript Object Notation (JSON) payload that carries a schedule serialized acceptably, at the transport level, even for the largest schedules exercised in the evaluation (Chapter 7). The machine and the solver configuration on which the algorithmic evaluation was conducted are reported with the evaluation protocol (Chapter 6).

5.2 The Template Engine

This section describes how the template abstraction of Section 4.3 is realized: how a template is stored, captured from an existing plan, instantiated onto a schedule, parameterized, and governed through a publication lifecycle.

5.2.1 Storage

A template is stored as a first-class entity, with its own tables and its own controllers, kept separate from those of a schedule even though a schedule instance and a template task carry similar fields. Each template carries a reference date, the base date, together with the hierarchy of its tasks and the *absolute* start and finish of every task as they were captured; a template is therefore a frozen calendar layout, a faithful snapshot of how the activities were once arranged in time. The reasons for storing absolute dates rather than relative offsets, and for keeping templates and schedules separate, are the design decisions set out in Section 4.3.1.

5.2.2 Capturing Templates from Existing Plans

Although a template can be created from scratch, in practice coordinators usually begin from the plans they already maintain. The capture flow lets a coordinator open an existing plan as an interactive timeline and select the activities that constitute a reusable group. Selection is hierarchical and works by drag and drop: dragging a summary task into the selection brings all of its descendants with it, together with the corresponding parent tasks in the hierarchy, and individual tasks can be added or removed one by one. The selected tasks need not form a single connected subtree; several disjoint groups can be gathered into the same template. On submission, the selected tasks are recreated as a new template, preserving their hierarchy and the dependencies among them, and

the base date of the template is set to the earliest start among them. Figure 5.1 shows the capture flow.

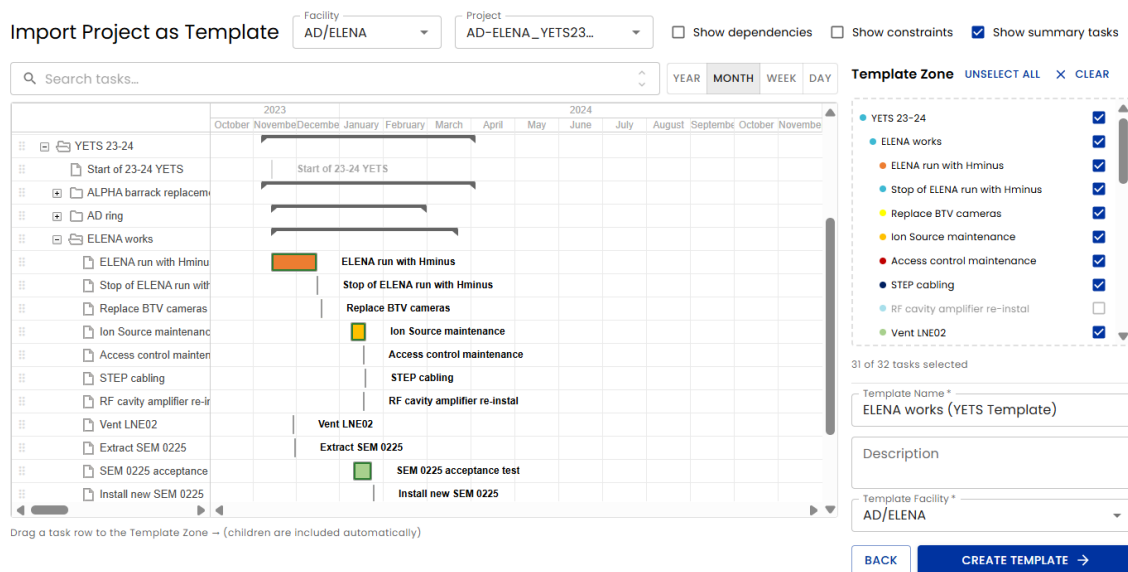


Figure 5.1: Capturing a reusable template from an existing plan.

5.2.3 Instantiation by Temporal Shift

To place a template into a schedule, all of its tasks are shifted to the requested start date. This shift is not a naive calendar translation, because most tasks consume only working time. A task that consumes working time is shifted with working-time arithmetic that respects the eight-hour working day, the five-day week, and the CERN holidays, so that a task that occupied two working days continues to occupy two working days after the shift, never straddling a weekend silently. A task that elapses continuously is shifted by a plain calendar translation, because such a task runs regardless of working hours. Groups, locations, dependencies, and scheduling constraints are copied along with the dates.

The arithmetic has subtle edge cases. When a task is shifted into a stretch of consecutive non-working days, its start and finish cannot be snapped to working time independently, or the finish may come to fall before the start; the finish is instead recomputed from the shifted start by adding the task's working-time duration, which advances correctly across the intervening weekends and holidays and preserves the duration.

One calendar, three implementations. The working-time calendar, that is, the eight-hour day, the five-day week, and the holiday list, is needed in three places: the backend uses it to shift tasks, the optimizer to schedule them, and the web client to validate manual edits and to flag violations as the coordinator works. Asking the backend for every such conversion would be too slow for an interactive interface, so the same calendar logic is implemented in each of the three. The three implementations must agree exactly, since a violation shown to the coordinator has to match the

schedule the optimizer produced. The replication is a maintenance cost, accepted consciously: the same arithmetic lives in three languages, and a change to it must be made three times. Two factors keep that cost contained. The calendar data itself is not replicated — all three implementations consume the same holiday list, so what is triplicated is arithmetic, not configuration — and the implementations are exercised against one another at the boundary cases, where defects of this kind were in fact found and fixed during development (Section 5.5).

5.2.4 Batch Instantiation

The same activity often has to be planned for a whole series of similar items, such as a sequence of magnets, so instantiation is also available in batch, as a feature in its own right. A single operation creates a chosen number of instances, advances the start of each by a configurable interval, optionally renames the tasks of each instance by a textual find-and-replace pattern that increments a trailing number or letter per instance, and optionally links consecutive instances by a dependency of a chosen type. Instances are independent by default and may overlap in time; the chaining option serializes them when the work is genuinely sequential. Two kinds of dependency therefore coexist in a composed schedule: the *intra-template* dependencies captured inside each template, and the *inter-instance* dependencies — equivalently, inter-template dependencies — that the coordinator adds between instances during composition, of which this chaining is a special case. Figure 5.2 shows both instantiation flows.

Batch renaming. The find-and-replace rename of batch instantiation splits the find pattern into a prefix and a trailing run, which is numeric by default and alphabetic when the starting value is a letter, and increments that run per instance: a pattern ending in “8” with a starting value of 9 produces 9, 10, 11, zero-padded to the width of the original, and a pattern ending in a letter with a starting value of “B” produces B, C, D. The replacement applies to every occurrence of the pattern within a name, so the dialog previews the resulting instance names and warns when the pattern occurs more than once in a name, letting the author catch an unintended substitution before it happens. The numbering scheme targets the common sequential case, in which instances differ only by a counter; when instances vary in more than a counter, such as in a location or an equipment name, the named placeholders of Section 4.3.2 carry that variation instead.

5.2.5 Implementing Parameterized Templates

The placeholder model of Section 4.3.2 is realized in the editor and stored relationally, as one row per placeholder, per reference, and per binding. Within the template, the author marks the precise points where a placeholder appears — a task whose location is the placeholder, a task whose team is the placeholder, or a fragment of a task name that the placeholder stands for — and each such mark is stored as a reference. At instantiation, the coordinator gives each placeholder a concrete value, and every reference to it is replaced by that value. The relational form keeps the historical bindings queryable, recording for every instance exactly which equipment and which

Facility

AD/ELENA

Template

AD ring (YETS)

Instance start date

08/11/2027 09:00

Must be on or after parent task start (2027-11-08 09:00)

☒ Enforce start date

Nest under task (optional)

ELENA works

Customize this instance (optional)

Pick a target equipment, override the instance name, or set substitutions. Leave collapsed to clone the template verbatim.

CANCEL

ADD

Batch Instantiate

Facility

AD/ELENA

Template

Power Cuts (EBD7)

How do you want to create instances?

BY RENAME PATTERN

BY SPECIFIC EQUIPMENT

Will create 3 instances (same date), chained FS

Power Cuts (EBD8) → Power Cuts (EBD9) → Power Cuts (EBD10)

Rename instances (optional) ?

Find text

EBD7

Text in the template name to find. Leave empty to keep the original name.

Start value

8

First number or letter used in replacement; increases by 1 per instance.

Count

3

Interval (days)

0

Days between instance starts

Start date

01/11/2027 08:00

Must be on or after schedule start (2027-11-01 08:00)

☒ Enforce start date

Chain instances

Finish-to-Start (FS)

Dependency type between consecutive instances

Preview (3 instances)

#	Name	Day offset
1	Power Cuts (EBD8)	+0d
2	Power Cuts (EBD9)	+0d
3	Power Cuts (EBD10)	+0d

2 FS dependency chains will be created

CANCEL

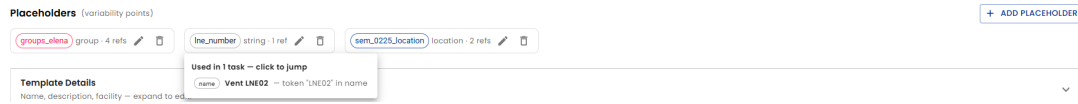
CREATE 3 INSTANCES

(a) Single instance

(b) Batch instantiation

Figure 5.2: Instantiating templates into a schedule, singly and in batch.

teams it was bound to; this is what realizes, at the data level, the programmatic integration point set out in Section 4.3.2. Figure 5.3 shows the two halves of the mechanism, declaring placeholders and binding them at instantiation, and Figure 5.4 shows the editor, declaring a placeholder and marking, task by task, the fields it stands for.



(a) The placeholders of a template, with the affordance for adding one

Add Template Instance

Facility
AD/ELENA

Template
Test Elena

Instance start date
01/11/2027 08:00

Must be on or after schedule start (2027-11-01 08:00).
Tasks will be shifted to start from this date.

☒ Enforce start date

Nest under task (optional)

Bindings (variability points) 3/3

*This template defines 3 variability points. Pick one value per point — the system fills it in across every task that references it. Click *Used in: ...* on any point to see exactly where it's applied.*

groups_elena group Used in: 6 tasks
Pick a group
EN-EL

lne_number string Used in: 1 task
Value
LNE03

Preview at instantiation:
"Vent LNE02" → "Vent LNE03"

sem_0225_location location Used in: 2 tasks
Pick a location
Ring

CANCEL ADD

(b) Binding placeholders at instantiation

Figure 5.3: Defining placeholders in a template and binding them at instantiation.

5.2.6 The Template Lifecycle

The publication lifecycle of Section 4.3.2 is realized as a state on each template that moves through draft, in review, approved, and archived, with the identity and time of an approval recorded. The

Add Placeholder

Name *

Lowercase letters, digits, underscores. Must start with a letter. Max 30 chars.

Type *

Location

At instantiation, the chosen location replaces (or is injected into) the location field of every task that references this placeholder.

Description (optional)
Shown to whoever instantiates the template.

CANCEL
CREATE

Variability on this task (variability points)

inject location (no source on this task) → sem_0225_location ×

inject group (no source on this task) → groups_elena ×

Mark which field as variable? ▼

CANCEL
MARK AS VARIABLE

(a) Declaring a placeholder

(b) Marking the variable fields of a task

Figure 5.4: Authoring a placeholder: declaring it on the template and marking, task by task, the fields it stands for.

instantiation pickers draw attention to the state of each template and exclude archived ones, so that schedules are normally composed from approved, stable templates. A single rule keeps this consistent: editing an approved template automatically returns it to draft, and the user is told that the approval has lapsed. The approved state therefore always corresponds to the exact content that was approved, never to content that was approved and then quietly changed.

5.3 The Optimizer

This section describes how the scheduling problem specified in Section 4.5 is encoded in CP-SAT: the variable model that reconciles the two kinds of time, the translation of constraint dates, the encoding of the concurrency limits, the search configuration, an optional two-phase warm start, and the diagnostic procedure behind the infeasibility explanations.

5.3.1 The Dual-Timeline Encoding

The two kinds of task identified in Section 4.5.1, regular and elapsed, force the model to reason about working time and continuously elapsing time on a common scale; reconciling them is the central problem in the optimizer’s model.

Why the two obvious encodings fail. Two single-domain encodings were considered and found inadequate. Encoding every task in calendar minutes would require, for each regular task, an explicit prohibition against every lunch break, every night, every weekend, and every holiday within its possible range; the number of such forbidden intervals grows with the horizon and the

number of regular tasks, inflating the model and the cost of constraint propagation well before the scale of a real schedule is reached. Encoding every task in working minutes, conversely, leaves the elapsed tasks without a natural representation, because a task that runs continuously does not align with a calendar made only of eight-hour working days.

Two domains bridged by a compact lookup. The adopted model uses two variable domains and a unified view over both. A regular task is represented by start, end, and interval variables in a *working-minute* domain, in which time advances only during working hours and the calendar irregularities simply do not exist. An elapsed task is represented by analogous variables in a *calendar-minute* domain. Every task, of either kind, additionally exposes a calendar-minute view of its start and end; for an elapsed task this view is its own variables, and for a regular task it is derived. The makespan and the cross-timeline dependencies are then expressed once, over the uniform calendar-minute view, while the bulk of the arithmetic for regular tasks stays in the fast working-minute domain.

The derivation of a regular task’s calendar-minute view from its working-minute variable is where the model earns its compactness. A direct table mapping each working minute to its calendar minute would have as many entries as there are working minutes in the horizon, which is prohibitive over a planning horizon of months. Instead, the working-minute value is decomposed into a working-day index and a minute-within-the-day, and each component is mapped through a small table with the solver’s `AddElement` constraint: one table maps a working-day index to a calendar-day index, skipping weekends and holidays, and a second table maps a working-minute-within-the-day to a calendar-minute-within-the-day, skipping the lunch break. The calendar minute is then recomposed from the two results (Fig. 5.5). The first table has one entry per day of the horizon and the second has one entry per working minute in a single day; both are therefore small and, crucially, independent of the number of tasks. The bridge is linear in the size of the schedule rather than proportional to the total working time it spans, which is what keeps the dual-timeline model compact as the schedule grows. Unlike the specialized calendar propagators developed for constraint solvers (Kreter et al., 2015), this formulation uses only standard constraints — two `AddElement` table lookups — so it requires no custom solver extension and remains portable, while accommodating both working-time and elapsing tasks in a single model.

When both endpoints of a dependency and its lag are in working time, the dependency is posted directly in the working-minute domain for efficiency, and the calendar view is used only when an elapsed quantity is involved.

5.3.2 Anchor-Relative Constraint Dates

A scheduling constraint that pins a task to an absolute calendar date interacts badly with inter-template dependencies. If one template must follow another, the whole instance has to slide later in time; but a constraint expressed as an absolute date would remain fixed and would pull against the slide, producing a spurious infeasibility. The constraint dates are therefore translated, inside the model, into offsets relative to the earliest task of their instance, so that when the instance slides

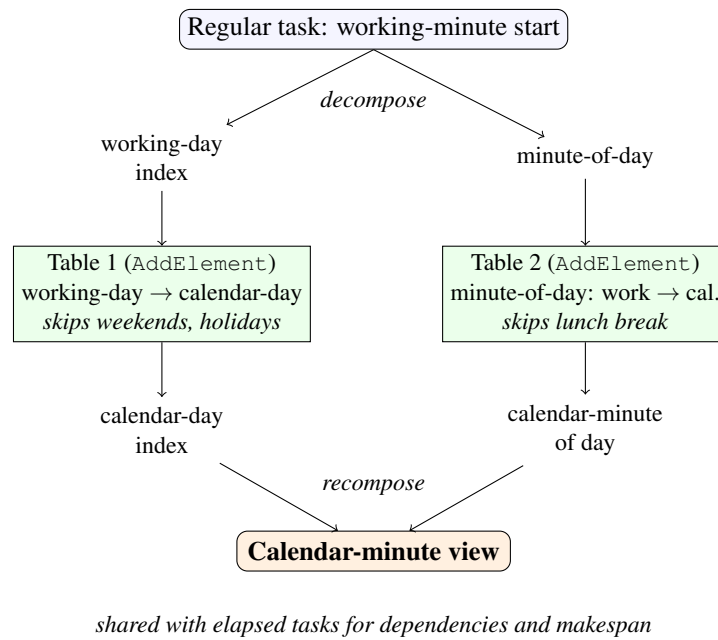


Figure 5.5: Bridging the working-minute and calendar-minute domains with two compact lookup tables.

as a whole the constraints slide with it. The absolute meaning of a constraint is thereby preserved relative to the actual work it governs, rather than relative to a calendar position that the optimizer is free to move.

Truly fixed dates remain available where they are wanted. A milestone is an ordinary task of zero duration and may carry the same constraints as any other task; a date that must not move under any circumstance is expressed as a lock, which pins the task absolutely and does not participate in the anchor-relative translation; and in a schedule without inter-instance dependencies the anchors themselves do not move, so the translated offsets coincide with the dates as captured.

5.3.3 Encoding the Concurrency Limits

The concurrency limits specified in Section 4.5.2 are posted over the task intervals of the dual-timeline model. A limit of one becomes a no-overlap constraint over the relevant intervals, and a higher limit a cumulative constraint with unit demand and the limit as capacity. For these limits alone, time is discretized into half-day slots, a resolution deliberately coarser than the one at which dependencies and dates are enforced. The choice rests on three considerations. The first is model size: enforcing the limits at full resolution would multiply the intervals over which the global constraints must reason, inflating the model well before realistic schedule sizes. The second is fit: a half-day is a natural unit for this kind of reasoning, since intervention work tends to be planned in blocks of mornings and afternoons rather than minute by minute. The third is safety: the discretization rounds up, so that a task using any part of a slot counts as occupying the whole slot, and the solver therefore never underestimates how many tasks run at the same time. The accepted price is occasional pessimism: two short tasks that share a slot without overlapping in time still

count as two. This coarse discretization applies only to the concurrency limits; dependencies and scheduling-constraint dates are enforced at full resolution.

5.3.4 Search Strategy and Solver Parameters

The choice of constraint programming is, in practice, the choice of CP-SAT's search. CP-SAT runs a portfolio of search strategies in parallel, each in its own worker, over a shared pool of solutions and bounds (Perron & Didier, 2025). The workers include a clause-driven satisfiability search that is effective at proving lower bounds and closing the optimality gap, a linear-relaxation worker that supplies strong dual bounds, one or more large neighborhood search workers that repeatedly destroy and rebuild parts of the incumbent solution and are the primary means of finding good schedules quickly, a core-based worker suited to contention among resources, and a feasibility-pump worker that produces an initial feasible solution to seed the others. Whenever any worker improves the incumbent or tightens a bound, all of the others see the improvement and prune accordingly.

Three parameters of the solve were fixed deliberately. The number of parallel workers is detected from the central processing unit (CPU) quota granted to the container at runtime, rather than from the host's core count; the detection matters because, under a container orchestrator, the naive core count reports the whole node rather than the slice allotted to the service, which would oversubscribe the processor. On top of the detection, the worker count is capped at twelve. The cap is the service's default configuration rather than a hardware fact: it was set so that a solve fits comfortably within the workstation used for development and evaluation while leaving headroom for the rest of the system, and it is a configuration parameter, so a deployment with a larger CPU allocation can raise it without a code change. Within the cap, more workers mean more distinct strategies explored simultaneously and thus a better chance of a quick, good solution; well beyond it, additional workers increasingly duplicate the portfolio's strategies rather than diversify them. The time resolution of the model is one working hour per solver unit, a configurable parameter; this default, and the effect of finer resolutions, is examined in the algorithmic evaluation (Chapter 7). Finally, the solve terminates in one of three ways: when CP-SAT proves the incumbent optimal, when the search has converged and continued effort no longer improves the schedule, or when a configurable wall-clock limit is reached (set to 900 seconds in the algorithmic evaluation), which is acceptable for long-term shutdown planning; a running solve can also be canceled from the interface, upon which a callback stops the search and the best schedule found so far is returned. When the search completes within these bounds, the returned schedule is proven optimal with respect to the discretized model; a finer discretization could, in principle, admit a shorter schedule. When the time limit intervenes first, the solver returns the best feasible schedule it has found, which may not be provably optimal.

A consequence of this portfolio search is that the optimizer is not deterministic. Repeated solves of the same schedule may follow different search paths and may report different solve times: several of the parallel workers are randomized, and the solver is run without a fixed random seed, so the order in which workers find solutions, and therefore the trajectory of the search,

varies between runs. This is the expected behavior of a modern parallel scheduling solver and is not a defect. It is also benign for the use case, because the quantity that coordinators care about, the makespan, is expected to remain stable across runs even when the solve time varies; the algorithmic evaluation quantifies this stability, including the variation across repeated runs (Chapter 7).

5.3.5 The Two-Phase Warm Start

The search space of a fine-grained model over a horizon of months is large, and a common way to tame it is to solve first a coarse, relaxed model and then to use its solution to guide the fine model. Such a two-phase scheme was implemented as an option. A first phase solves the schedule at day-level, or optionally week-level, granularity, using critical-path lower bounds along the dependency chains, and produces an approximate placement for every task; the second phase then uses that placement both as a warm start and as the center of a tightened domain window around each task.

The two-phase warm start is a configurable capability, and in the default configuration it is disabled, so that the production solve is single-phase at one-hour resolution. The effect of the warm start on solve time and on schedule quality, and the reasoning behind this default, are examined in the algorithmic evaluation (Chapter 7).

5.3.6 Infeasibility Diagnostics

Two complementary mechanisms produce the explanations required by Section 4.5.4. A set of static checks runs before every solve, in negligible time, and catches the impossibilities that need no search to detect: a constraint date implausibly far from the plan, a lock that contradicts a scheduling constraint on the same task, or a duration that cannot fit before a must-finish date. When the full solver does return infeasible, a diagnostic procedure isolates the cause by building a sequence of reduced models of increasing scope, starting from the bare durations and adding, in turn, the locks, the intra-template dependencies, the scheduling constraints, and the inter-template dependencies; the first scope at which the reduced model becomes infeasible identifies the family of constraints responsible. Dependency cycles are found within this procedure by a strongly connected components decomposition, and a constraint that conflicts with a chain of predecessors is identified by a critical-path deficit computation that needs no further search (Tarjan, 1972).

Each diagnosis is reported as a typed violation with a suggested one-click fix; Table 5.2 lists the violation typologies the procedure distinguishes, how each is detected, and the fix offered for it. When the procedure cannot isolate a single culprit, for instance because the conflict involves the interaction of several resource limits, it reports that explicitly and asks the coordinator to review the case, rather than proposing a destructive change on a guess. Applying a selected set of fixes re-runs the optimization, so that the coordinator converges on a feasible schedule through a short sequence of explained, reversible steps (Fig. 5.6).

Table 5.2: Violation typologies identified by the infeasibility diagnostics, their detection, and the suggested one-click fix.

Violation	Detection	Suggested fix
Lock contradicting a date constraint on the same task	static check	unlock the task or relax the constraint
Duration that cannot fit before a must-finish date	static check	relax the constraint or shorten the task
Dependency cycle	strongly connected components	remove an edge of the cycle
Date constraint against a predecessor chain	critical-path deficit	relax the constraint
Over-long inter-instance lag	reduced models	remove or shorten the offending edge
Conflicting constraint family	first failing scope in the sequence of reduced models	review the constraints of that family
Cause not isolable (e.g. interacting resource limits)	—	reported explicitly for manual review

❗ Resolve schedule conflicts

The optimizer could not produce a feasible schedule. 1 issue detected. Select the fixes you want to apply, then re-run.

☒
ERROR
DEPENDENCY_CYCLE
Fix: REMOVE_DEPENDENCY

Intra-template dependency cycle between 2 task(s): Vent LNE02, Extract SEM 0225. Pick which dependency to remove to break the cycle.

Affected tasks: Vent LNE02 Extract SEM 0225

Will remove 2 dependencies:

🔗 Vent LNE02 —FS→ Extract SEM 0225

🔗 Extract SEM 0225 —FS→ Vent LNE02

CANCEL

APPLY 1 FIX & RE-RUN

Figure 5.6: The conflict resolver: structured violations, each with a suggested one-click fix.

5.4 The Interactive Interface

The web client realizes the visualization design of Section 4.6 and the review workflow of Section 4.7 by extending the DHTMLX Gantt component already used by the Scheduling Tools platform. Figure 5.7 shows the assembled schedule page: the task grid and the enriched timeline

occupy the center, under a toolbar that toggles each visual layer and sets the time scale, while the side panel holds, from top to bottom, the list of instances, the inter-instance dependencies, the concurrency limits, the optimization launcher with its result summary and change list, and the export.

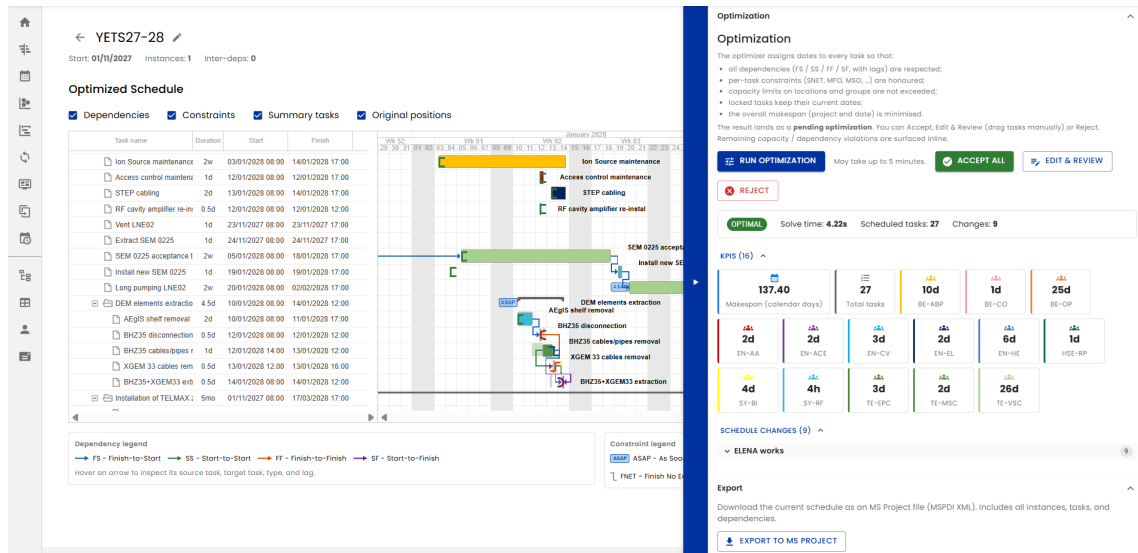


Figure 5.7: The schedule page as a whole: the enriched timeline at the center, with the composition, optimization, and export controls in the side panel.

The constraint glyphs, the dependency arrows, the lock markers, and the before-and-after overlays are drawn as custom layers above the task bars; Figure 5.8 shows the realized notation with its legend. The concurrency heatmap is computed in the web client directly from the task intervals, as the peak number of concurrent tasks per day for each limited resource, and is shown in Figure 5.9.

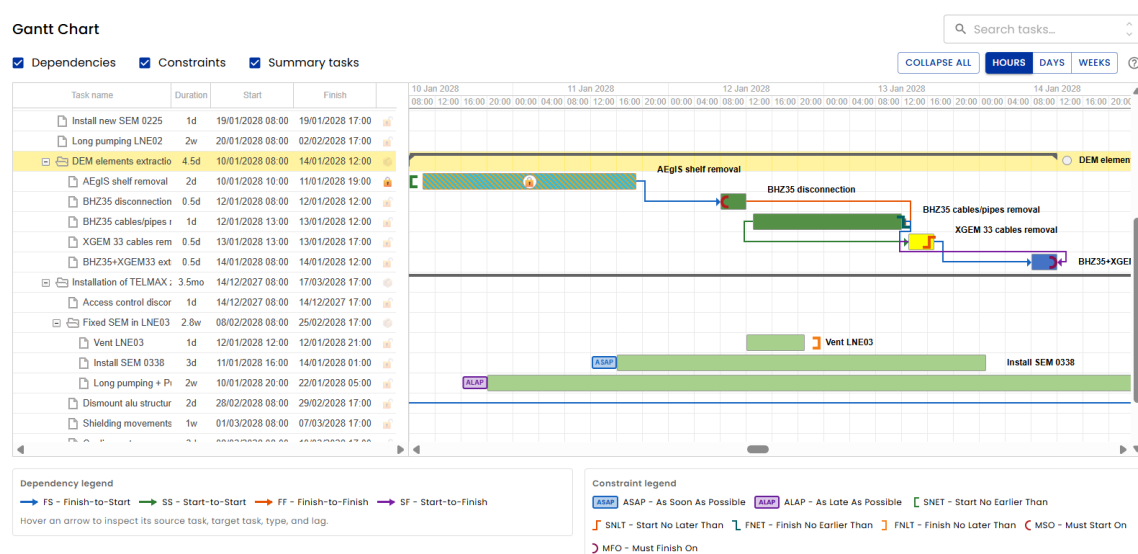


Figure 5.8: The visual notation for scheduling constraints and dependencies, with its legend.

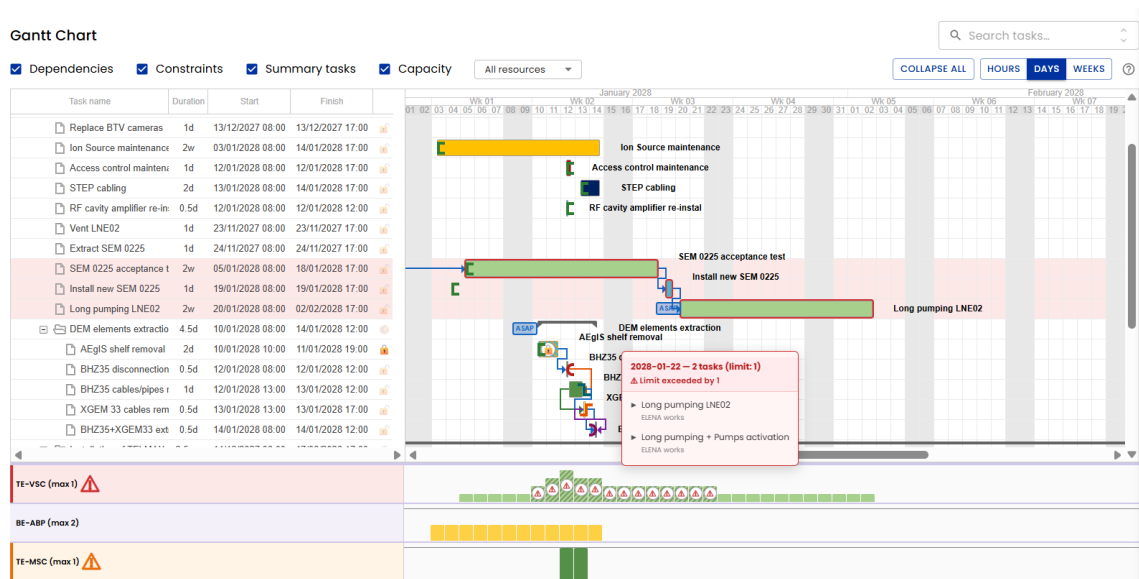


Figure 5.9: The concurrency heatmap beneath the timeline: a violated limit (red), a limit exactly reached (orange), and a respected limit, with the per-day detail shown on demand.

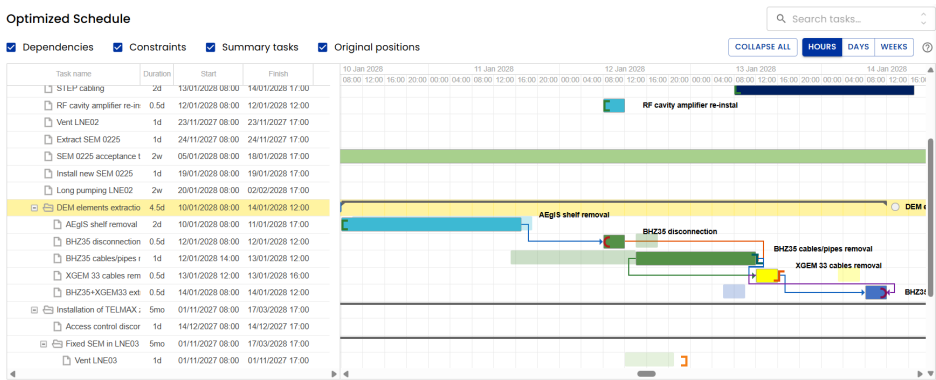
The before-and-after overlay draws the pre-solve position of each task as a faint ghost bar behind the optimized bar, paired with the reason-annotated change list (Figure 5.10). In the review mode, manual edits are validated live against the client-side implementation of the working-time calendar (Section 5.2.3), so that dependency and concurrency violations appear as the coordinator drags a task, without a round trip to the backend (Figure 5.11).

5.4.1 Persisting and Adjudicating Runs

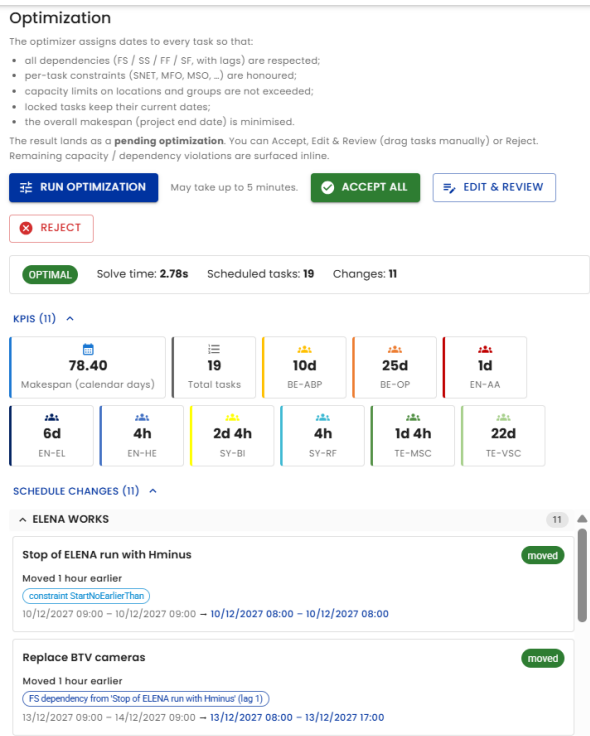
The decision workflow of Section 4.7 is realized by persisting every solve as a *run*: a record that stores both the input that produced it and the output it returned, together with the decision status that carries it from pending to accepted or rejected and, thereafter, possibly to reverted. Storing the input beside the output is what makes a revert cheap: the dates that preceded an accepted run are recovered from its stored input, so that an undo requires no separate snapshot. The single-pending rule is enforced on this record — at most one run may be pending for a schedule at any time — which removes an entire class of stale-state ambiguity from the interface; any edit that would invalidate a pending decision prompts the coordinator to confirm before it proceeds.

5.4.2 Export to Microsoft Project

A finalized schedule is exported to Microsoft Project, so that the surrounding CERN tooling, which consumes Microsoft Project files, can ingest it. The export covers the schedule and its instances only, never the templates themselves: it assembles every instance as a sub-project under one master file, preserving the task hierarchy, and carries, for each task, the dates, the scheduling constraint, the lock state, the team, and the planning contact, together with the intra-template and

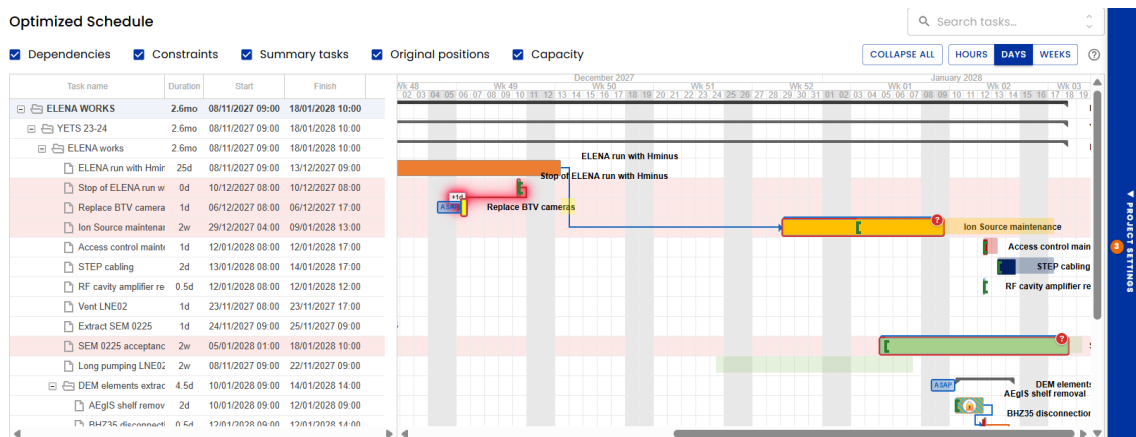


(a) The optimized timeline with the original positions shown as overlays

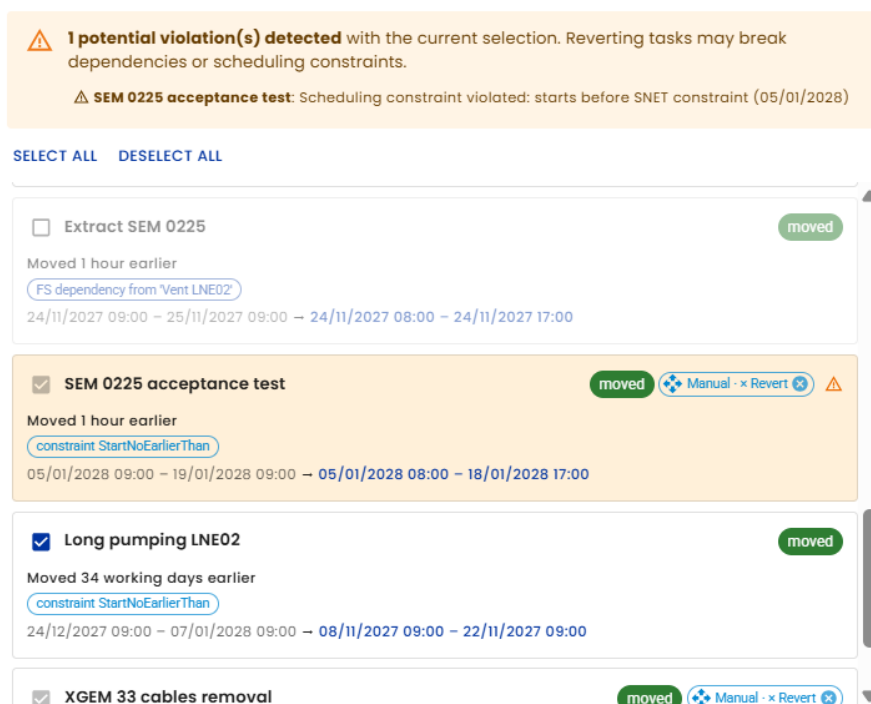


(b) The optimization panel: status, KPIs, and the reason-annotated change list

Figure 5.10: The enriched timeline: the before/after ghost overlay and the reason-annotated change list.



(a) Manual overrides and live violations on the timeline



(b) The change panel: per-change acceptance and live violation feedback

Figure 5.11: Reviewing an optimization: partial acceptance, manual override, and live violation feedback.

inter-instance dependencies and their lags. The file is written through the MPXJ library (Iles, 2019).

This export is the practical reason for two earlier decisions. It is why instance dates are concrete calendar dates at every point of the workflow, since the round trip to and from Microsoft Project is expressed in absolute dates (Section 4.3.1); and it is the file-based counterpart of the programmatic integration that the named placeholders enable (Section 4.3.2). It is also what closes the loop with the existing toolchain (Section 4.1): a composed schedule, exported back to Microsoft Project, re-enters the same ecosystem from which its templates were captured, rather than ending in a separate application that stands apart from it. Figure 5.12 shows an exported schedule opened in Microsoft Project.

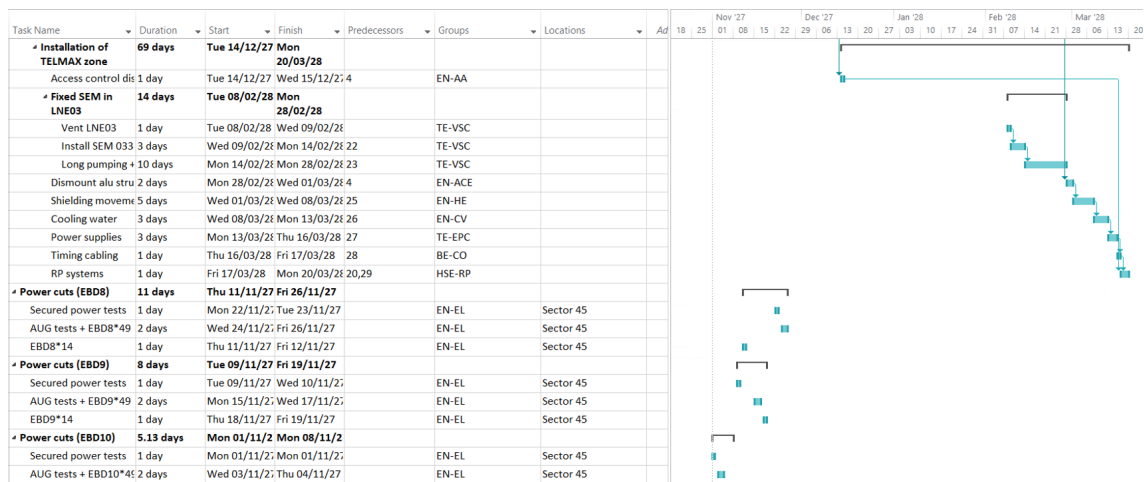


Figure 5.12: An exported schedule opened in Microsoft Project.

5.5 Challenges

A number of challenges arose while building the prototype, of which the most demanding are described here. The most demanding was the representation of time inside the optimizer. Regular tasks advance only during working hours, whereas elapsed tasks run continuously, so neither a pure calendar model nor a pure working-time model could express both at once, and the two had to coexist in a single solver. The arrangement that works, a pair of variable domains joined by two small lookup tables (Section 5.3.1), is not the obvious one: the natural first attempt, a single table mapping each working minute to its calendar minute, is unworkable because the table grows with the length of the schedule, and only the decomposition into two compact tables keeps the model small. A related subtlety appeared once instances could depend on one another. A dependency can push an entire instance later in time, but a constraint tied to a fixed calendar date does not move with it, and the two end up pulling the affected task in opposite directions. Constraint dates are therefore expressed relative to the work they govern rather than as fixed calendar positions (Section 5.3.2).

Making infeasibility intelligible was a second challenge. A constraint solver reports only that a model has no solution, which tells a coordinator facing dozens of interacting rules nothing about what to change. Producing a usable explanation meant isolating a likely cause without simply re-solving the model: a sequence of progressively larger sub-models, a strongly-connected-components search for dependency cycles, and a critical-path calculation for conflicts between a constraint and a chain of predecessors (Section 5.3.6). The procedure is unavoidably heuristic, and when it cannot single out one culprit it reports that explicitly rather than proposing a change on a guess.

A third challenge was less deep but more persistent: keeping the working-time calendar consistent across the three places that use it. The backend shifts tasks with it, the optimizer schedules with it, and the web client validates the coordinator's edits with it, and a violation flagged in the browser must agree, to the minute, with the schedule the optimizer produced. The arithmetic is therefore implemented three times, in three languages, and kept in step by sharing the same holiday data and by testing the implementations against one another (Section 5.2); small discrepancies in it, especially around weekends and holidays, were a recurring source of defects during development. Comparable care was needed at the boundary with Microsoft Project, which reschedules dates of its own accord when a file is opened and could otherwise discard the schedule the optimizer had computed (Section 5.4.2).

5.6 Summary

This chapter described how the approach was realized as an extension of the CERN Scheduling Tools platform. The template engine realizes the storage, capture, instantiation, parameterization, and lifecycle of templates, reproducing the same working-time calendar across the three components that depend on it (Section 5.2); the optimizer runs as a separate constraint-programming service whose dual-timeline encoding maps working time onto calendar dates (Section 5.3); and the interactive interface enriches the platform's existing Gantt component with the constraint notation, the concurrency heatmap, and the before-and-after overlay, and exports the finalized schedule back to Microsoft Project (Section 5.4). The three challenges that most shaped the engineering, the replicated calendar, the infeasibility diagnostics, and the boundary with Microsoft Project, are the practical counterpart of the design decisions of Chapter 4. The evaluation of the result is the subject of the next two chapters.

Chapter 6

Evaluation Protocol

This chapter presents the protocol for evaluating the planning tool specified in Chapter 4 and realized in Chapter 5. The prototype combines three distinct components: a reusable *template engine*, a constraint-programming *optimizer*, and an interactive *review and export interface*. Consequently, a single evaluation strategy is not sufficient to address all of them with the appropriate rigor. The protocol is therefore organized into three complementary evaluation tracks, each targeting a different facet of the research problem and using an appropriate methodology.

The chapter first presents the evaluation objectives and the research questions they address. It then describes the three tracks together, covering for each its rationale, its procedure, and its methodology, along with the end-to-end case study that complements them. The remaining sections detail the participants and the ethical safeguards, the apparatus and instruments, the task scenarios and metrics with the data analysis plan, and the pilot study. The protocol is presented as designed before data collection; its execution, including any deviation from this plan, is reported with the results in Chapter 7.

6.1 Objectives and Research Questions

The evaluation is formative: it is designed to produce evidence that informs and refines each component on its own terms, as well as feedback that can guide the prototype’s continued development. Four objectives guide the whole protocol:

- **O1 — Validate the template abstraction.** Determine whether the proposed abstraction of a *template* is expressive enough to represent the recurring activities that CERN coordinators deal with in practice, and whether users can extract, edit, and reuse such templates after only a short introduction.
- **O2 — Validate the optimizer.** Characterize the behavior of the CP-SAT optimizer across a representative range of problem sizes and constraint densities, establish the effect of the main solver parameters, and establish whether the solver returns feasible and interpretable schedules on instances built from real CERN templates.

- **O3 — Assess the usability of the prototype.** Evaluate the usability of the template engine, the visualization choices, and the Edit & Review mode, in particular, how effectively they support coordinators in composing, understanding, and refining schedules.
- **O4 — Evaluate the workflow benefit.** Assess coordinators' perceived gain in time and effort relative to the current manual MS Project workflow, and the conditions under which they would adopt the tool.

The protocol is aligned with the two research questions introduced in Chapter 1: RQ1 asks what visual representation techniques and 'visual language' principles can be employed to model, display, and interact with complex scheduling constraints in task and project planning systems, and RQ2 asks how constraint-rich project scheduling in this context can be modeled and solved, and what trade-offs between schedule quality and computational cost the resulting approach exhibits.

The template engine contributes to both research questions simultaneously: as a visual and interactive technique for expressing recurring groups of scheduling constraints (RQ1), and as the structural input on which the optimizer operates at solve time (RQ2). RQ2 is addressed in two steps: the analysis of optimization families against the problem's characteristics (Section 3.3.4) motivates a constraint-programming formulation, and the algorithmic benchmark (Track C) then characterizes the trade-offs of the resulting model across problem sizes and solver parameters.

6.2 Evaluation Design

The evaluation is organized into three tracks, executed in parallel but collecting independent data sets. Track A and Track C each target the research question for which their methodology is best suited: Track A studies the features and the usability of the prototype with users, and Track C benchmarks the optimizer offline. Track B assesses the workflow benefit (objective O4), a cross-cutting concern whose comparison bears on the integrated tool and therefore complements both research questions; it is administered in the same session as Track A, which is why the two share their participants. Together the tracks provide a balanced evidence base. This section presents, for each track, its design, its procedure, and the methodological rationale behind it.

6.2.1 Track A: Features and Usability

Track A is a qualitative, in-person, moderated study in which coordinators and other CERN members execute a scripted set of tasks that exercise the main features of the prototype. The data collected are task completion, time-on-task, error counts, think-aloud commentary, a System Usability Scale (SUS) questionnaire, and open-ended feedback. Its primary research question is **RQ1**.

Each Track A session follows a fixed sequence; the indicated durations are planned budgets:

1. **Consent and demographics (3 min).** The participant reads and signs the informed consent form (Appendix A.2) and completes the demographic questionnaire (Appendix A.3).

2. **Tool introduction (3 min).** The moderator introduces the tool with a short scripted walk-through (Appendix A.1) covering the Templates and Schedules pages, the Gantt legend, and the optimization panel buttons.
3. **In-test tasks (32 min).** The participant executes the eight tasks listed in Section 6.5; full instructions and expected outcomes are reproduced in Appendix A.4. The participant is asked to think aloud during each task.
4. **Task ratings (3 min).** After completing all tasks, the participant rates each of the eight tasks on two 5-point Likert scales (subjective difficulty and perceived usefulness) using a single consolidated grid (Appendix A.5), with one optional free-text field at the end for any additional comment.
5. **SUS questionnaire (4 min).** The standard ten SUS items (Brooke, 1996) are administered via the web form (Appendix A.6).
6. **Open-ended interview (5 min).** The moderator asks five open questions (reproduced in Appendix A.6) on the perceived usefulness of the tool, the weakest parts of the experience, suggestions for improvement, the aspect of the tool that surprised the participant the most (positively or negatively), and any missing features.

The session is designed to fit within approximately 50 minutes. If the participant is a planning coordinator, the session continues with the Track B workflow comparison questionnaire described next, for an additional 10 minutes, bringing the full session to approximately 60 minutes.

The in-person moderated format is a deliberate methodological choice. It allows the moderator to observe participant hesitation, collect think-aloud commentary, and pose clarifying questions in real time, and direct observation is important to confirm that participants engage correctly with the intended features and to understand the reasoning behind their actions. While the in-person requirement limits the total number of participants that can be recruited within the evaluation window, it supports high-quality qualitative data and a controlled environment.

6.2.2 Track B: Comparative Workflow Questionnaire

Track B is a structured questionnaire administered at the end of each Track A session, exclusively to planning coordinators. Participants are asked to compare the workflow experienced in the prototype with their current workflow in Microsoft Project. The questionnaire covers confidence in the prototype's output, perceived speed, behavior under increasing schedule complexity, the aspects of the current workflow most and least likely to benefit, willingness to adopt, and the conditions for full replacement of the current workflow. Its primary objective is **O4** (workflow benefit).

Immediately following the Track A open-ended interview, the moderator guides the coordinator through the questions, reading each aloud and recording the responses on their own laptop. This procedure requires approximately ten minutes and does not involve any additional hands-on interaction with the prototype or with Microsoft Project. A fully objective comparison, in which

each participant would rebuild the same plan in both Microsoft Project and the prototype, was not feasible within the session budget of approximately one hour; Track B therefore elicits the coordinators’ perceived comparison rather than a measured one, a limitation revisited in Chapter 7.

6.2.3 Track C: Algorithmic Benchmark

Track C is a quantitative offline study that benchmarks the CP-SAT optimizer on instances built from real CERN templates. Independent variables fall into instance-level factors (number of instances, template, inter-template dependencies, concurrency tightness) and solver-level factors (Phase 1 warm-start and granularity). Dependent variables include solve time, makespan, optimality gap, and feasibility rate. Its primary research question is **RQ2**.

Track C is executed offline and does not involve participants. Its procedure is:

1. **Benchmark case preparation.** A set of benchmark cases is constructed from real CERN data. Schedules of different sizes are generated by adjusting the number of template instances and the templates selected. Inter-template dependencies and capacity limits are incorporated at the schedule level to regulate overall density.
2. **Configuration sweep.** For each benchmark case, the benchmarking harness runs the optimizer under each combination of the independent variables defined in Section 6.5.4. Each combination is repeated at least three times to account for the stochastic elements of the CP-SAT search.
3. **Data logging.** Each run writes a comma-separated values (CSV) row with solver status, wall-clock time, solve time reported by the solver, number of tasks, makespan in calendar days, optimality gap, and any error message. The request and the result are archived as JSON.

Where Tracks A and B focus on the human side of the tool, Track C requires no participant intervention: a dedicated testing harness measures the optimizer’s performance across a wide range of problem scales, so this part of the evaluation is objective and reproducible, complementing the user-facing tracks.

Real-instance case study. In addition to the three tracks, the approach is validated end-to-end on one real shutdown, the PS Year-End Technical Stop 2025–26, a plan of 492 leaf tasks spanning 70.7 calendar days. The study measures how much of the real plan the template abstraction covers — the fraction of its tasks that captured templates represent — and how faithfully the optimizer reproduces the real plan, comparing both the overall duration and the per-task date agreement; it also characterizes where, and why, the optimized and the real schedules diverge. This provides the external reference point that the parameter sweep of Track C, run on instances built from templates, cannot supply. The results are reported in Chapter 7.

6.3 Participants, Ethics, and Moderation

Two participant populations are recruited for the tracks involving user tests:

- **Planning coordinators.** Members of the EN-ACE-OSI section and of neighboring engineering groups who regularly use MS Project to plan technical stops. These participants anchor the external validity of Tracks A and B: their judgment on whether the template abstraction is expressive enough and whether the workflow reduces their effort is the most informative data point. Recruitment is done through the supervisor, Fernando Pedrosa, who has standing contact with the relevant teams.
- **Non-planning CERN members.** CERN personnel who are not involved in the planning process but possess domain knowledge regarding accelerator shutdowns, teams, and locations. Their participation is limited to Track A and serves to assess the intuitiveness of the interface for users lacking prior mental models of the current MS Project workflow.

A total of 20 participants are recruited: 14 planning coordinators and 6 non-planning CERN members. Participants are required to be CERN members in order to preserve the relevance of the domain context during the sessions and to simplify the ethical and data-handling requirements. All sessions are conducted in person at CERN between May 4th and May 21st, 2026.

Ethical considerations. Each participant reads and signs an informed consent form (reproduced in Appendix A.2) before any data are collected. The form describes the purpose of the study, the types of data collected, and the participant’s right to withdraw at any moment without consequence. All data are anonymized at the point of collection: each participant is assigned a numeric identifier used consistently across all artifacts, with the mapping held privately by the researcher. Screen or voice recordings are captured only with the participant’s explicit consent. Participation is voluntary, and participants may interrupt the session at any time without consequence.

Moderation. The researcher acts as the sole moderator for all in-person sessions. The moderator is responsible for scheduling the session, preparing the environment, reading the consent form with the participant, introducing the prototype, guiding the participant through the tasks, observing and taking notes, and conducting the post-test interview. To reduce demand characteristics, the moderator follows a fixed script during the introduction and during task presentation, and does not provide task-specific help beyond the items pre-agreed with the supervisor. All additional hints or interventions are logged so that they can be factored into the analysis.

6.4 Apparatus and Instruments

Track A and Track B sessions are held at the EN-ACE-OSI premises at CERN in a quiet meeting room. Each participant uses the moderator’s laptop. Track C is executed on a development workstation, an Intel Core i7-12700H with fourteen cores and twenty threads and sixteen gigabytes of

memory, kept connected to mains power throughout, with the solver at its default twelve-worker cap (Section 5.3.4); running-time observations are only comparable when the hardware is specified, which is why it is fixed here.

The following instruments and tools are used across the three tracks:

- **Prototype.** OSS Coordination tool (Scheduling Tools), configured with the full feature set used in the evaluation and connected to the CERN database.
- **Microsoft Project.** A recent, locally-installed version, used to open the file exported from the prototype in task T8 and to verify that tasks, dependencies, and constraints are preserved.
- **Questionnaire platform.** A web form with three sections: a pre-test questionnaire collecting demographics, self-rated familiarity with the CERN planning workflow and with the AD/ELENA facility — the Antiproton Decelerator (AD) and its associated Extra Low Energy Antiproton ring (ELENA) — on 5-point Likert scales (the latter used to control for prior facility knowledge, since all in-test tasks operate on AD/ELENA projects), and the planning tools currently used; the consolidated task-ratings grid (Section 6.5.1); and a post-test questionnaire with the ten SUS items (Brooke, 1996) followed by five open-ended questions. The Track B questionnaire (Appendix A.7) is administered orally by the moderator, who records the responses directly on their own laptop.
- **Benchmarking harness (Track C).** A Python driver that submits a batch of configurations to the optimizer, logs solver status, wall-clock time, and objective values, and produces a single CSV for analysis. The harness uses the same HTTP endpoints as the backend so that the measurements reflect production behavior.

6.5 Task Scenarios and Metrics

This section lists the eight in-test task scenarios and their intended evaluation targets. The scenarios are chosen so that each main feature of the prototype is exercised by at least one task and so that each research question is addressed by more than one task (see Section 6.5.5). The tasks are ordered to follow the natural user journey: the participant first learns about the template engine, then creates and enriches a schedule, runs the optimizer, and finally reviews and exports the result. Task durations are indicative planned budgets and are refined after the pilot study (Section 6.6).

T1 — Open and inspect a template (3 min). The participant opens an existing template and inspects both the Gantt view and the table view, switching between the two. They identify the summary tasks and the leaf tasks, read the duration and the responsible group of one specific leaf task, and identify one intra-template dependency by hovering over the link. Evaluates the legibility of both the Gantt and the table representations of the template, and the dependency visual language.

T2 — Create a template from an existing project (3 min). The participant imports a project from the database, locates the summary task of interest, drags it to include its entire subtree, and deselects one specific task that should not be included. They name the template and confirm. Evaluates the drag and deselect affordances of the import-from-project flow.

T3 — Edit a template (6 min). The participant opens the template created in T2. They rename one task, change a duration, delete one task, and set a Start-No-Earlier-Than (SNET) constraint on one leaf task using template-relative day numbering (the template editor expresses dates as offsets from the template start rather than as absolute calendar dates). They then use undo to revert the last action and redo it, to verify the undo/redo stack. They add a finish-to-start dependency between two tasks. Finally, they assign a group to one task. Evaluates the template editor, the constraint-type dialog and its visual representation in the Gantt, the undo/redo interaction, and the group assignment.

T4 — Create a schedule and instantiate templates (4 min). The participant creates a new schedule with a specified start date in a target facility. They first add a single instance of a template (the one they created in T2). They then use the batch instantiation dialog to add three instances of a template, applying a find-and-replace substitution that renames the instances and the corresponding task names from a numeric pattern. Evaluates both the single and the batch instantiation flows and the find-and-replace substitution on task names.

T5 — Add an inter-instance dependency and capacity limits (4 min). Using the optimization panel, the participant adds a finish-to-start dependency between two instances (instance-to-instance). They then configure two capacity limits, applied across the whole schedule: a location concurrency limit of one on the relevant CERN sector, and a group concurrency limit of one on the responsible team. Finally, they identify a pre-optimization capacity violation that is already present on the schedule. Evaluates the inter-dependency editor, the constraints editor, the handling of both location and group concurrency constraints, and the visualization of capacity violations.

T6 — Lock a task, run the optimizer, and interpret the result (5 min). The participant first locks one specific task so that the optimizer cannot change its dates. They then run the optimizer. When the result appears, they inspect the change list, the KPI cards, and the ghost overlay; they also check whether the location and group concurrency limits set in T5 are respected in the optimized schedule. They explain aloud what changed, which tasks moved, and what caused the main movement. Evaluates the lock toggle, the optimization trigger, the visualization of the output, and the participant's ability to reason about the optimizer's decisions.

T7 — Edit & Review the result (4 min). The participant enters Edit & Review mode. They manually drag a task on the Gantt to a position that violates a finish-to-start dependency, observe the live violation alert that appears in the changes panel, and resolve the violation by reverting the

manual drag through the per-change revert affordance (clicking the *X* on the *Manual* badge in the expanded changes list). Evaluates the Edit & Review mode, the manual-override interaction in the Gantt, the violation feedback loop, and the per-change revert affordance.

T8 — Accept and export (3 min). The participant saves the accepted result of the Edit & Review session and exports the schedule to Microsoft Project. They then open the exported file locally and confirm that the tasks, their dependencies, and the constraints are preserved in Microsoft Project. Evaluates the closure of the user journey.

6.5.1 Task Ratings

After completing all eight in-test tasks, the participant fills a consolidated grid rating each task on two 5-point Likert scales (subjective difficulty, 1 = very easy, 5 = very difficult; perceived usefulness of the exercised feature, 1 = not useful, 5 = very useful) plus a single optional free-text field at the end for any additional comment. The full grid is reproduced in Appendix A.5.

6.5.2 Performance Metrics (Objective)

For each task the following objective metrics are collected:

- **Completion rate.** Binary: the participant produced the expected outcome of the task. The per-task durations of Section 6.5 are session-pacing budgets that bound the total session length, not pass-or-fail thresholds; running past one is captured by the time-on-task metric rather than counted as a failed completion.
- **Time-on-task.** Duration from task presentation to completion, measured by the moderator.
- **Error count.** Number of unsuccessful attempts that the participant had to recover from.
- **Assists.** Number of hints or clarifications granted by the moderator beyond the scripted introduction.

6.5.3 Perception Metrics (Subjective)

The subjective metrics aggregate as:

- **Per-task difficulty and usefulness.** Two 5-point Likert ratings per task, reported as distributions rather than means.
- **SUS score.** The standard SUS score in [0, 100], computed as in (Brooke, 1996) and interpreted against established empirical benchmarks rather than ad-hoc cut-offs (Bangor et al., 2008; Lewis & Sauro, 2009); the benchmark values themselves are presented alongside the results (Chapter 7).

- **Thematic analysis of open-ended responses.** The free-text answers to the post-test questions are coded inductively to identify recurring themes (e.g. weakest parts of the experience, most-valued features, missing functionality).
- **Track B workflow comparison ratings.** Responses to the three 5-point Likert questions (confidence in output quality, perceived speed, and behavior under complexity), reported as distributions. Free-text responses to the four open-ended questions are coded using the same inductive thematic analysis applied to Track A responses.

6.5.4 Algorithmic Metrics (Track C)

Track C defines its own metrics, which are independent of user perception:

- **Solve wall-clock time.** Time between the HTTP request to `/optimize` and the HTTP response, measured by the benchmarking harness.
- **Solver status.** `OPTIMAL`, `FEASIBLE`, `INFEASIBLE`, or `ERROR`, as reported by CP-SAT.
- **Makespan.** Duration of the optimized schedule, in calendar days.
- **Optimality gap.** The relative gap between the best solution found and the solver’s proven lower bound, when reported.
- **Feasibility rate.** The fraction of runs that return `OPTIMAL` or `FEASIBLE` within the timeout.

The independent variables of Track C fall into two groups, instance-level and solver-level; Table 6.1 lists each variable and the values tested. Each template instance comprises roughly one hundred leaf tasks, so the benchmarked schedules range from about two hundred leaf tasks at $N = 2$ to roughly two thousand at $N = 20$.

Table 6.1: Independent variables of the Track C benchmark and the values tested.

Group	Variable	Values tested
Instance	Number of instances N	2, 4, 6, 8, 10, 12, 16, 20
Instance	Template	two real CERN templates of contrasting density
Instance	Inter-instance dependency structure	none; a growing number of random links; finish-to-start chaining
Instance	Concurrency limits (location and team, separately and combined)	1, 2, 3, 5
Solver	Phase 1 warm-start mode	off; day-level; week-level
Solver	Granularity (<code>SOLVER_GRANULARITY</code>)	15, 30, 60 minutes
Solver	Timeout	fixed at 900 s

6.5.5 Mapping Tasks to Research Questions

Table 6.2 summarizes how each task contributes to the two research questions. A single ✓ denotes a contributing task; ✓✓ denotes an essential piece of evidence for that research question.

Table 6.2: Mapping of in-test tasks and algorithmic experiments to the research questions.

Track	Task / Experiment	RQ1	RQ2
A	T1 Inspect template (Gantt + table)	✓	
A	T2 Template from project	✓✓	
A	T3 Edit template	✓✓	
A	T4 Schedule + batch instantiate	✓✓	
A	T5 Inter-instance dep + limits	✓✓	✓
A	T6 Lock, optimize, interpret	✓✓	✓
A	T7 Edit & Review	✓✓	
A	T8 Accept and export	✓	
B	Workflow comparison questionnaire	✓	✓
C	Instance count sweep (N)		✓✓
C	Template (dependency density)		✓✓
C	Inter-instance dependency structure		✓✓
C	Concurrency tightness		✓✓
C	Phase 1 warm-start (off/day/week)		✓✓
C	Solver granularity		✓✓

6.5.6 Data Analysis Plan

The data collected by the three tracks are analyzed with methods appropriate to each.

Track A. Objective metrics for each task, such as completion rate, time-on-task, error count, and number of assists, are summarized using means, medians, and standard deviations. Subjective Likert ratings of difficulty and usefulness are reported as distributions rather than means, given the ordinal nature of the scale (Jamieson, 2004).

The System Usability Scale (SUS) score is calculated following the standard procedure (Brooke, 1996) and interpreted based on established empirical thresholds (Bangor et al., 2008; Lewis & Sauro, 2009). Responses to open-ended questions and think-aloud commentary are analyzed using inductive thematic analysis (Braun & Clarke, 2006) to identify recurring themes. Given the small and unbalanced sample (14 coordinators, 6 non-planning members), the analysis is primarily descriptive and qualitative; any group-level or correlational comparison, such as between coordinators and non-planning members or against years of Microsoft Project experience, is reported only as exploratory, hypothesis-generating observation, without inferential significance claims.

Track B. Responses to the three Likert-scaled questions are reported as distributions. Free-text responses to the four open-ended questions are coded using inductive thematic analysis (Braun & Clarke, 2006), with a particular focus on perceived improvements, perceived limitations, and conditions for adoption.

Track C. Because the CP-SAT search is non-deterministic, every configuration is solved three times and the three replicates are reported individually rather than collapsed into a single average, so that the run-to-run spread — and therefore the stability of each result — is visible directly. Solve time and makespan are plotted against problem size (the number of template instances, and equivalently the total leaf-task count), to characterize how each grows as the schedule scales. For every configuration the feasibility rate (the fraction of replicates returning a feasible or optimal schedule within the 900-second limit), the solver status, and the optimality gap are reported; runs that reach the time limit are recorded as censored rather than as a precise solve time, so that the growth in solve time is not understated. Particular attention is paid to the makespan: although solve time varies between replicates, a makespan that is stable across the three runs confirms that the quantity coordinators ultimately rely on is reproducible even when the solver does not prove optimality within the limit.

6.6 Pilot Study

A pilot session is conducted with a single colleague from the EN-ACE-OSI section on May 4th, 2026, before the main data collection begins. The pilot executes the full Track A procedure and the Track B workflow comparison questionnaire in full. The pilot pursues three goals: to verify that the session fits within the time budget, to detect ambiguous wording in the task statements and in the questionnaire, and to detect feature-level defects that would bias the observation.

All adjustments resulting from the pilot, including changes to task wording, order, time budgets, or procedures, are documented in Chapter 7. This allows readers to trace the instrument's development from its initial version to the one used in the main study.

If the pilot leads to significant changes in the procedure or multiple task statements, a second pilot is conducted to confirm that the revised instrument functions as intended. If adjustments are limited to wording, the main study begins immediately and the pilot session is retained as the first of the twenty recorded sessions.

6.7 Summary

This chapter has presented the evaluation protocol for the planning tool, organized as three complementary tracks. Track A studies features and usability through a qualitative moderated session with CERN coordinators and other members. Track B collects coordinators' perceptions of the workflow benefit through a comparison questionnaire at the end of the Track A session. Track C

benchmarks the optimizer on instances built from real CERN templates across a controlled set of parameters.

The three tracks collect evidence for the two research questions of Chapter 1: visualization and interaction (RQ1) and optimization (RQ2). The data analysis plan combines descriptive statistics, with group comparisons treated as exploratory, and a thematic analysis of the open-ended responses. A pilot study verifies the instrument and procedure before the main data collection. The next chapter reports the results and their implications for each research question.

Chapter 7

Results

This chapter reports the outcomes of the evaluation specified in Chapter 6. It follows the order of the three evaluation tracks: it first characterizes the participant sample and the usability results of Track A, then presents the comparative workflow questionnaire of Track B and the algorithmic benchmark of Track C, and finally the end-to-end case study on a real programmed stop. The limitations that qualify these results are then examined. To preserve anonymity, participants are identified throughout by unique IDs, and their roles are reported at the level of function rather than their exact position.

7.1 Sample Characterization

Twenty CERN members took part in the study, fourteen planning coordinators and six members in non-planning roles, matching the recruitment plan of Section 6.3. The pilot session led to no change in the task wording, order, time budgets, or procedure; in accordance with the rule established in Section 6.6, no second pilot was conducted, the main study proceeded directly, and the pilot session was retained as the first of the twenty recorded sessions.

The participants spanned a wide range of ages and experience. Ages ranged from 25 to 62 years, and the sample comprised fifteen men and five women. Self-reported experience with Microsoft Project was markedly bimodal, ranging from half a year to thirty-five years, with a median of four and a half years and a mean of nine that reflects a tail of long-standing schedulers alongside several recent adopters. Familiarity with the CERN planning workflow averaged 4.0 on the five-point scale, whereas familiarity with the AD/ELENA facility used in the in-test tasks was uniformly low. This placed the participants on an equal footing, working on a domain unfamiliar to most rather than one they already knew. Figure 7.1 summarizes the age and experience distributions; the complete per-participant demographics are tabulated in Appendix B.1 (Table B.1), together with the planning tools each participant currently uses (Table B.2).

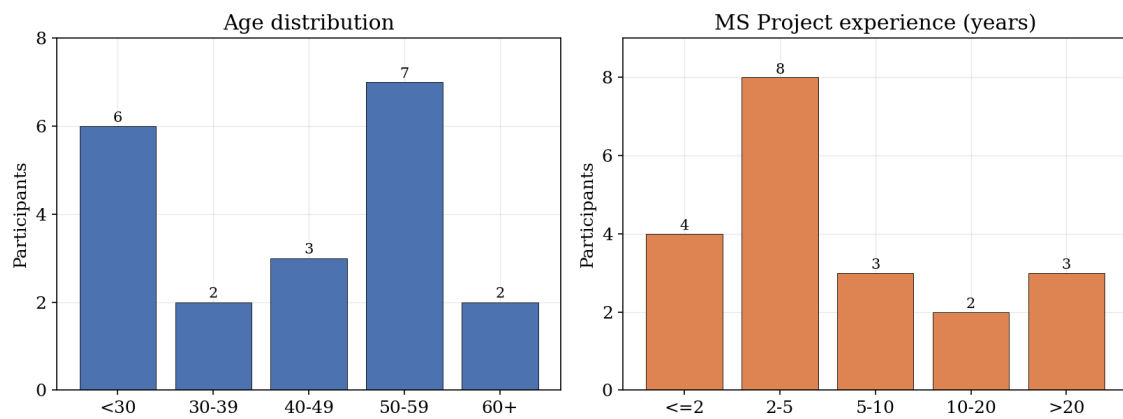


Figure 7.1: Age distribution (left) and self-reported Microsoft Project experience (right) of the twenty participants.

7.2 Track A: Usability Results

Track A combined objective operability metrics, a standardized usability questionnaire, and open-ended feedback. The objective metrics are reported first, followed by the perceived task ratings, the System Usability Scale, and the qualitative feedback.

7.2.1 Task Performance

Every participant completed every one of the eight tasks. Because the tasks were guided, scripted walkthroughs (Section 6.5), a full completion rate was expected. If a participant struggled, the moderator offered a hint or pointed out the error so that the session could continue. The planned task durations of Section 6.5 served as session-pacing budgets rather than pass-or-fail thresholds; running past one was captured by the time-on-task metric, not counted as a failed completion. Therefore, the completion rate is reported mainly to confirm that no task contained a blocking defect. The more informative signals of operability are the time spent on each task, the errors made, and the moderator assists requested.

Figure 7.2 shows the distribution of time on task. The most time-consuming tasks were editing a template (T3) and creating a schedule with batch instantiation (T4), with the highest medians, at 4.0 and 4.2 minutes; these two tasks bundle several sub-operations, which account for their longer duration, and their times also vary more across participants than those of the short, linear tasks. The two quickest tasks were the first and the last, opening and inspecting a template (T1) and exporting the schedule (T8), each completed in a median of under two minutes. This mirrors a deliberate ordering of the script, which opens with a light inspection task to let participants settle in and places the more demanding interactions in the middle, rather than confronting participants with the hardest tasks at the very start. The per-participant times are tabulated in Appendix B.3 (Table B.3).

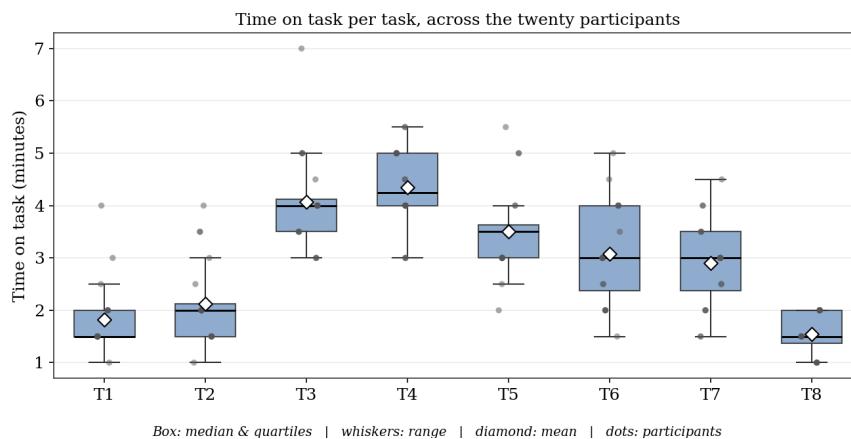


Figure 7.2: Distribution of time on task across the twenty participants. Boxes show the median and quartiles, whiskers the range, the diamond the mean, and the dots the individual participants.

The planned task budgets of Section 6.5 offer a second reference for these times. Figure 7.3 sets each budget against the mean time actually spent. The budgets were deliberately generous—set high so that a fixed, time-boxed session would not overrun, rather than as best-guess predictions of how long each task would take—and they proved conservative accordingly: every task was completed, on average, within its allotted time, the sole exception being the creation of a schedule by instantiation (T4), which ran marginally over its budget, at 4.3 against 4.0 minutes. Across the eight tasks the total mean time was 23.4 minutes against a planned 32, which confirms that the session budget was comfortable and locates the only genuine time pressure in the instantiation step, the same interaction that the difficulty ratings single out in Section 7.2.2.

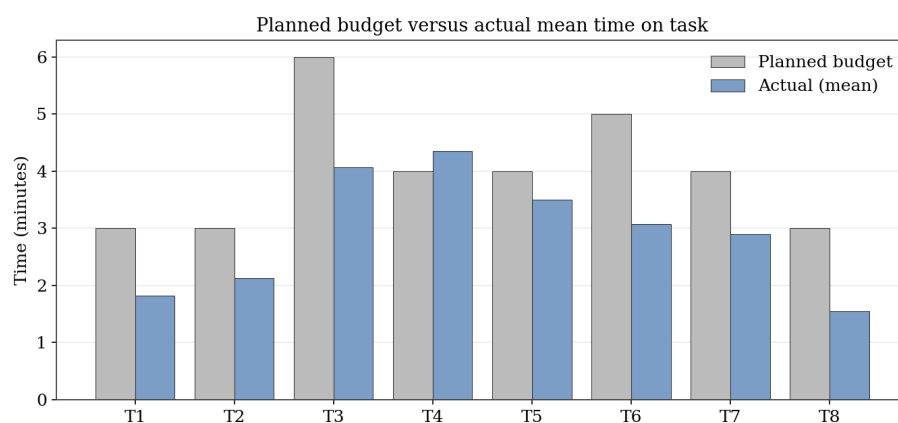


Figure 7.3: Planned budget against actual mean time on task, for tasks T1 to T8.

The errors and assists follow the same pattern as the time. Figure 7.4 reports the total number of errors and moderator assists per task. Both were low throughout, and both concentrate on the two longest tasks, T3 and T4: the tasks that took longest are also those on which participants made the most errors and requested the most help, whereas the short, linear flows of opening a template (T1) and exporting (T8) were essentially error-free. The minor difficulties thus cluster on template

editing and instantiation, the richest and least linear interactions in the workflow, which points to those two flows as the natural focus for further interface refinement.

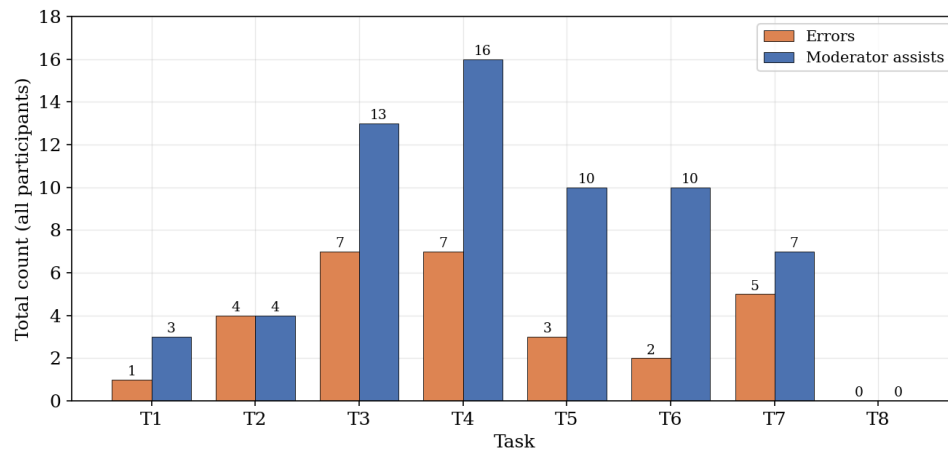


Figure 7.4: Total errors and moderator assists per task, across all twenty participants.

The per-task totals of Fig. 7.4 do not reveal how these difficulties were spread across participants. Figure 7.5 therefore reports the errors and assists summed over the eight tasks for each participant. The distribution is markedly skewed: most participants made one or two errors at most and requested little assistance, while a small number account for a disproportionate share of both. The difficulties identified above were thus concentrated in a few sessions rather than being a systematic obstacle for the sample as a whole.

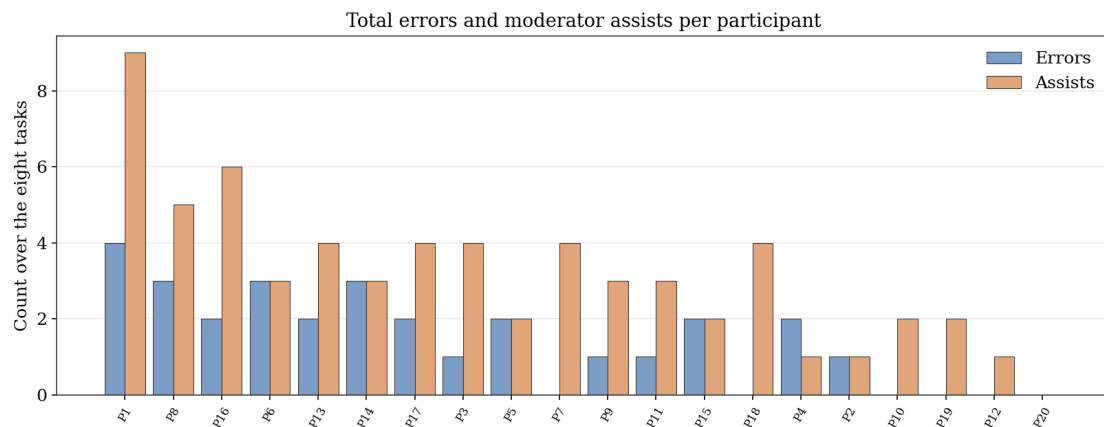


Figure 7.5: Total errors and moderator assists per participant, summed over the eight tasks.

7.2.2 Perceived Difficulty and Usefulness

After completing the tasks, participants rated each one on two five-point scales, for subjective difficulty and for the usefulness of the exercised feature (Section 6.5.1). Figure 7.6 shows the distributions. The two scales tell a consistent story: every task was rated easy and every task was

rated highly useful. The tasks perceived as marginally harder, creating a schedule (T4) and the Edit & Review step (T7), were also among the most novel interactions; even so, their difficulty medians stayed at two, with only a few participants rating them higher. The pairing of low perceived difficulty with high perceived usefulness indicates that participants found the features not only approachable but also worth the effort. The per-participant ratings are tabulated in Appendix B.3 (Table B.4).

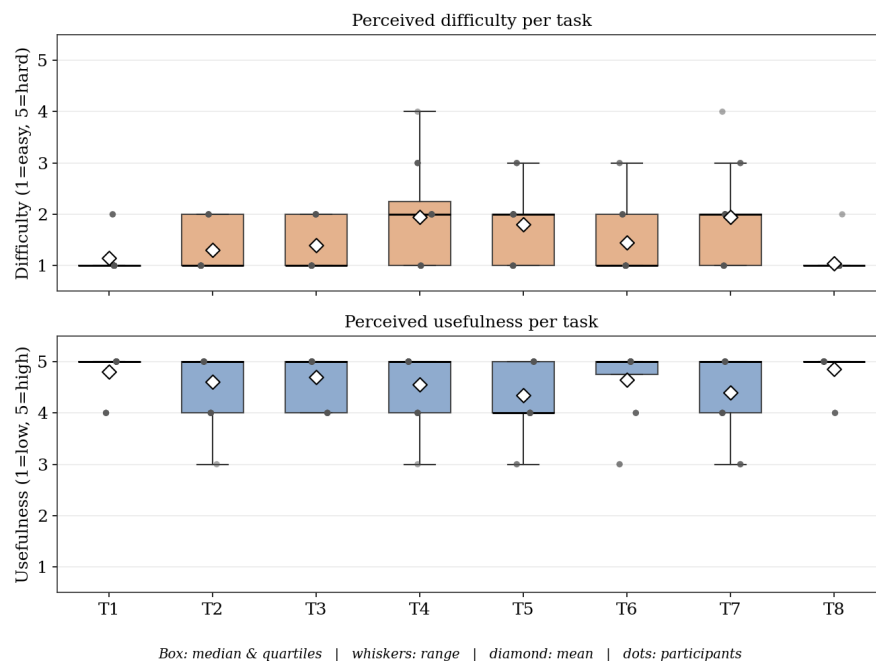


Figure 7.6: Distribution of perceived difficulty (top) and usefulness (bottom) for each task, on five-point scales. Boxes show the median and quartiles, whiskers the range, the diamond the mean, and the dots the individual participants.

7.2.3 System Usability Scale

The System Usability Scale (Brooke, 1996) yielded a mean score of 81.5 across the twenty participants. Figure 7.7 shows the score of each participant. Interpreted against the established benchmarks, this places the prototype well above the average: a score in the low eighties places the prototype in grade A and the upper quartile of systems in the percentile ranking of Lewis and Sauro (2009), and on the adjective scale of Bangor et al. (2008) it falls between the “good” and “excellent” ratings, closer to the latter, against an accepted industry mean of 68.

The full per-item SUS responses are reproduced in Appendix B.4 (Table B.5). Whether this score, and the operability metrics reported above, depended on the participant’s profile is examined next.

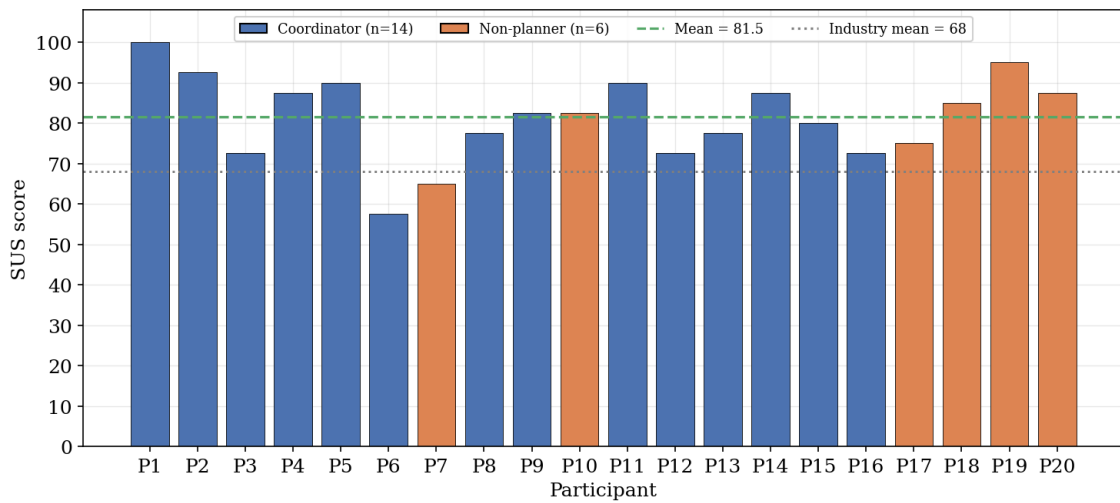


Figure 7.7: System Usability Scale score for each participant. The dashed line marks the sample mean of 81.5; the dotted line marks the accepted industry mean of 68.

7.2.4 Influence of Participant Background

A formative study with twenty participants cannot support strong inferential claims, but the sample is varied enough to ask whether the results above depend on who the participant was. Three contrasts are examined, along the dimensions on which the sample varies most: planning coordinators against members in non-planning roles, younger participants against older ones, and newcomers to Microsoft Project against long-standing users. Each metric is tested both with a Mann–Whitney U test between groups and, for the two continuous attributes, age and Microsoft Project experience, with a Spearman rank correlation that uses all twenty participants rather than dichotomizing them. The groups are small and the comparisons are exploratory (Section 6.5.6), so no correction for multiple comparisons is applied and the resulting p -values are read as indicative rather than confirmatory.

The System Usability Scale was independent of background on every contrast. Figure 7.8 shows the score distributions, which overlap almost completely: coordinators and non-planning members ($p = 1.00$), younger and older participants ($p = 0.63$), and newcomers and veterans ($p = 1.00$) are statistically indistinguishable. The rank correlations agree, with both age and experience essentially uncorrelated with the score ($\rho = -0.04$ and $\rho = -0.02$). The perceived usability of the interface therefore did not depend on prior familiarity with planning, on the participant's age, or on their Microsoft Project experience, which supports its intended accessibility to occasional as well as expert users.

The operability metrics tell a more nuanced story. Figure 7.9 contrasts the total time and errors across the same groups. Time on task was the metric that varied most: coordinators took longer than non-planning members (medians 25.8 against 21.0 minutes, $p = 0.04$) and older participants longer than younger ones ($p = 0.02$), and the rank correlation confirms a moderate association between age and time ($\rho = +0.48$, $p = 0.03$); the same upward trend with Microsoft Project

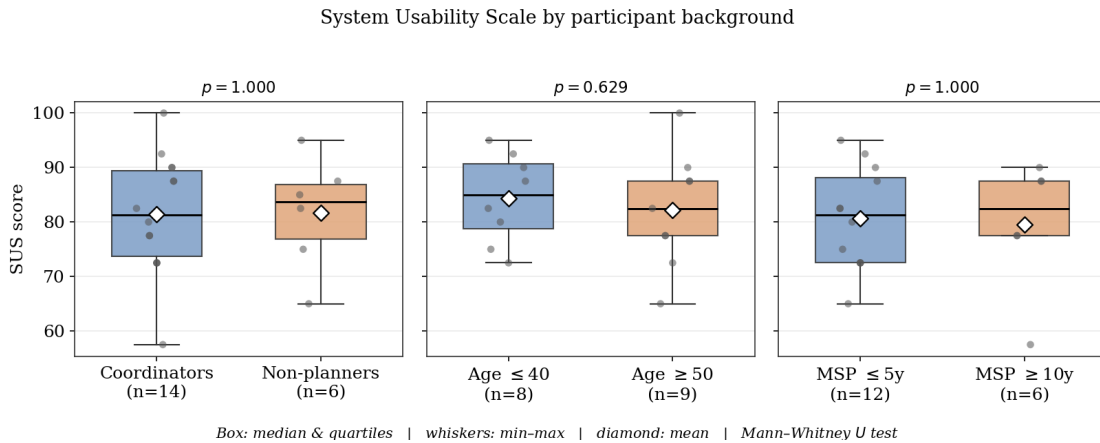


Figure 7.8: Distribution of SUS scores by participant group, for the three background contrasts. Boxes show the median and quartiles, whiskers the range, and the diamond the mean; the annotation gives the Mann–Whitney p -value.

experience was present but did not reach significance ($p = 0.08$, $\rho = +0.36$). Errors followed a similar but weaker pattern, with only the coordinator contrast significant (medians 2.0 against 0.0, $p = 0.01$). Assists, perceived difficulty, and perceived usefulness did not differ on any contrast.

A consistent reading emerges. The interface was equally usable and equally well received regardless of the participant’s profile, the result that matters most for its intended audience. Where the groups differed was in how they worked through the tasks: the more experienced and the older participants spent longer and made slightly more missteps, behavior consistent with a more deliberate, exploratory engagement with an unfamiliar tool rather than with any barrier to use, since it left both their task success and their satisfaction unaffected. These observations are bounded by the size and balance of the sample and characterize the present cohort rather than a generalizable effect.

7.2.5 Qualitative Feedback

The open-ended responses and the think-aloud commentary were analyzed thematically; the resulting themes are summarized in Appendix B.5 (Table B.6). They were strongly positive about the two core features, the template engine and the optimizer, while converging on a small number of concrete suggestions for refinement.

The templates and the optimization were, by a clear margin, the features that participants singled out as most valuable. Several described the optimization as an unexpected benefit, noting that it could improve not only the initial plan but also the efficiency of revising a schedule, and one observed that the graphical representation of the constraints was “surprisingly clear”. The ease of first use was raised repeatedly and without prompting; one user found the tool “remarkably easy to handle on first use, which is unusual for scheduling tools”, and several others also highlighted the speed of importing an existing project and the ability to instantiate a template into many copies in a single action.

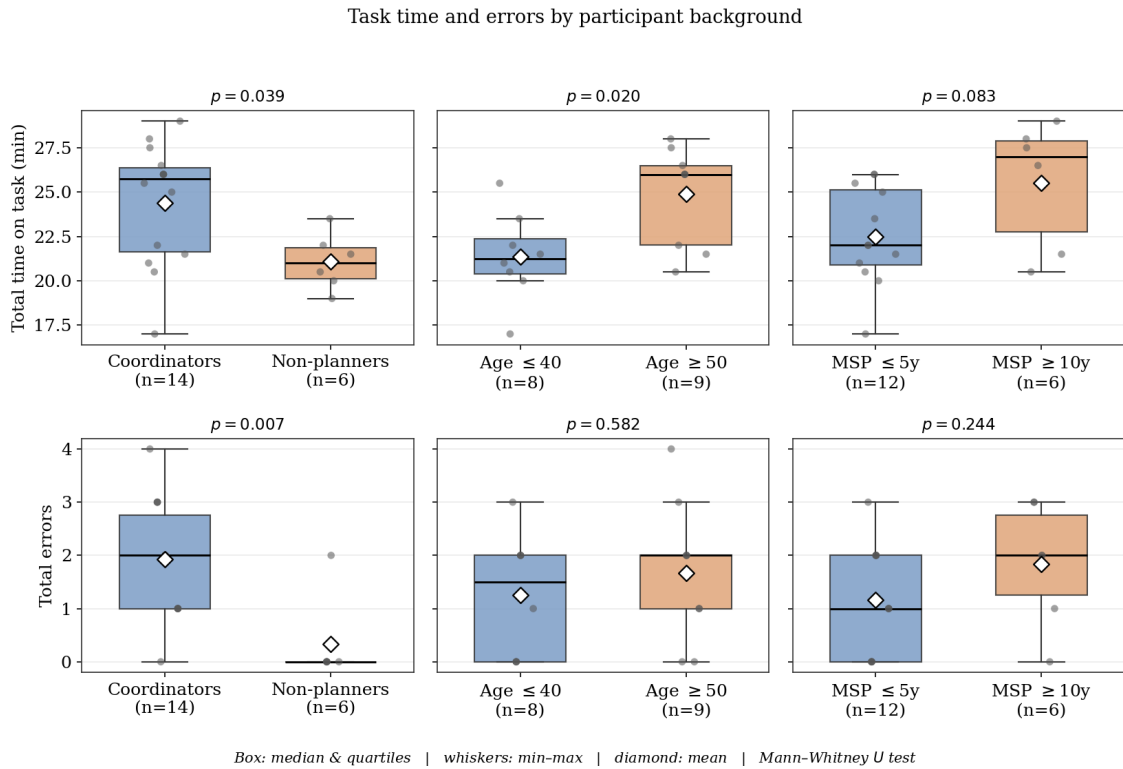


Figure 7.9: Total time on task (top) and total errors (bottom) by participant group, for the three background contrasts. Box elements and annotation as in Fig. 7.8.

The suggestions for improvement clustered around a few themes. Most frequently, participants asked for greater transparency around the optimization step: a side-by-side view of the schedule before and after optimization, and an indication of the solver’s progress while it runs. A second group of comments concerned the schedule-level editing controls, with several participants wanting undo and redo within the schedule view, as already available in the template editor, and clearer signposting of the accept-and-revert workflow. Others asked for a broader overview of the Gantt chart, with zoom and a critical-path view, and for export formats beyond Microsoft Project XML, such as native .mpp, Excel, or PDF.

The most experienced participants shared an honest insight: while templates capture standard, repetitive maintenance well, interventions with high variability in teams, working modes, or access conditions are harder to capture in a single reusable template. This observation is consistent with the case-study findings reported later in this chapter and with the coordinators’ own assessment in Track B, and it delineates the portion of the planning problem for which the template-and-optimize approach is best suited.

7.3 Track B: Workflow Comparison

The protocol planned the workflow comparison for every coordinator, at the end of their Track A session (Section 6.2.2). In practice, seven of the fourteen coordinators completed it: for the re-

mainder, the session had exhausted its time budget or the coordinator was not available for the additional ten minutes, and the questionnaire was skipped rather than rushed. The seven respondents are identified here as B1 to B7, and the adoption findings below are read against this response rate. Three questions used a five-point scale and four were open-ended; the responses are reproduced in Appendix B.6 (Tables B.7 and B.8).

On the three scaled questions, the coordinators were confident in the quality of the prototype’s output, with a mean confidence of 4.1, that it could produce a schedule at least as good as the one they would build by hand. They expected the prototype to be faster than their current workflow (mean 3.7) and, on balance, expected its benefit to hold or grow as schedules become more complex (mean 3.3), although this last question drew the widest spread of opinion, with one coordinator (B2) expecting the benefit to diminish.

The open-ended answers were consistent across respondents. Asked which aspects of their work the prototype would most improve, the coordinators pointed to the templates and the initial setup of a plan: the templates “avoid copy-paste errors in activity names and other manual mistakes” (B1) and remove the need to “open and copy from past projects” (B4), while the optimization “provides a global view of constraint compliance, especially for large shutdowns” (B3). Asked where the prototype would help least, the coordinators converged, almost unanimously, on the optimization of non-standard work: the optimizer “requires strict use of explicit constraints and dependencies; without the planner’s implicit systems knowledge, automatically moving tasks can produce unexpected results” (B1), and several noted that implicit or partial constraints — knowledge that coordinators carry as tacit experience — are difficult to formalize (B5, B6).

All seven coordinators said they would consider adopting the prototype for a real intervention plan: four without qualification, and three scoping their answer to small projects or to specific checks as a first step. When asked what they would need before replacing Microsoft Project entirely, most pointed not to a missing feature of the tool but to the maturity of its content: a complete, validated library of templates for each facility (B1, B2, B4, B7). Two acknowledged the breadth of features that decades of Microsoft Project development provide (B5, B6), and one considered the prototype already sufficient (B3).

These results align closely with the case study reported later in this chapter: the coordinators’ enthusiasm for the templates, and their reservation about the optimizer’s handling of implicit knowledge, are exactly the strength and the boundary that the case study quantifies on a real plan.

7.4 Track C: Algorithmic Results

Track C characterizes the behavior of the constraint-programming optimizer under controlled conditions, quantifying the trade-offs between schedule quality and computational cost. The setup is described first, followed by the four aspects of solver behavior examined: scalability with problem size, the effect of time granularity, the value of the warm-start, and the cost of concurrency limits. The complete per-configuration results are tabulated in Appendix B.7.

7.4.1 Benchmark Setup

The benchmark was run on two real CERN templates, built from tasks taken from past stop plans, that differ only in their internal dependency density. Both contain the same 97 leaf tasks per instance, with identical durations and resource requirements; Template 1 is densely constrained (85 intra-template dependencies, a density of 0.88 dependencies per leaf task), whereas Template 2 is sparse (density 0.43). Holding the task structure fixed while varying only the dependency graph isolates the effect of constraint density on solver behavior. Problem size was scaled by instantiating each template between 2 and 20 times, spanning 194 to 1940 leaf tasks. These sizes are anchored to the scale of a real stop: a Year-End Technical Stop runs up to around 1,500 leaf tasks — the case study’s stop, at 492, sits within that range — so the benchmark spans from a fraction of a single stop to beyond its upper end, while Long Shutdowns, larger again at thousands of tasks and occasionally tens of thousands, lie above the range exercised here. Around this, four factors were varied one at a time: the inter-instance dependency structure (none, or a chain linking consecutive instances), the time granularity, the Phase 1 warm-start mode, and the concurrency limits. Each factor was varied around a common base setting — 60-minute granularity, a day-level warm-start hint, no concurrency limits, and no inter-instance dependencies — except where the factor under study dictates otherwise. Each of the 64 resulting configurations was solved three times per template, for a total of 384 valid runs; the few launches that aborted on a transient infrastructure error (for instance a gateway timeout or a dropped connection, unrelated to the solver) were re-executed, so every configuration contributes three valid runs, each ending in an optimal, a feasible, or a timed-out state.

All runs used Google OR-Tools CP-SAT v9.14 with a 900-second timeout and the solver’s default twelve-worker cap (Section 5.3.4), on an Intel Core i7-12700H workstation with 16 gigabytes (GB) of memory kept connected to mains power throughout, so that no power-management throttling could affect the timings. In the figures that follow, each of the three runs of a configuration is drawn as a separate marker: a filled marker denotes an optimal solution, a hollow marker a feasible solution whose optimality was not proven within the limit, and a cross a run that reached the time limit without a solution.

7.4.2 Scalability

This experiment asks one practical question: does the approach scale to the size of a real programmed stop, and where does its tractability ceiling lie? To answer it in a controlled way, a single template is replicated to grow the task count while its internal structure is held fixed, so that only size and inter-instance coupling vary. Replicating one template is a deliberate abstraction — it isolates the effect of scale and constraint density rather than reproducing the heterogeneity of a real plan, which the case study of Section 7.5 supplies instead — and it is what lets the ceiling and the density effect reported below be attributed to size and coupling alone.

Figure 7.10 shows how solve time and makespan scale with problem size. On the instances that are solved, solve time grows roughly linearly with the number of tasks, and it does not run to

the full time limit: the solver stops early once it has proven the schedule optimal or the search has converged (Section 5.3.4), returning the best one found; on these templates that point was reached in under 300 seconds. The binding case for scaling is the absence of inter-instance dependencies: the instances are then fully independent, the search space is largest, and the solver reaches the time limit without a solution beyond roughly 1200 tasks — the two largest sizes, 1552 and 1940 tasks, did not close on either template. Introducing a chain of dependencies between consecutive instances changes the picture markedly. The chain is a finish-to-start dependency, under which each instance can begin only after the previous one has finished; this breaks the symmetry between otherwise interchangeable instances and shrinks the search, so the solver scales to the full 1940 tasks within the budget. Within this scalability experiment, a feasible or optimal schedule was returned in 87.5% of runs, with every failure at the two largest no-dependency sizes; across the full benchmark of 384 runs, the rate was 95.3%, and the sparser Template 2, whose effective search space is larger, was marginally harder than the denser Template 1 (93.8% against 96.9% across all experiments).

This rate can be decomposed by the strength of the result. Of the 384 runs, 72% returned a proven optimum, 23% a feasible schedule whose optimality was not proven within the budget, and the remaining 5% reached the time limit with no solution at all. The genuine failures fell exclusively at the largest no-dependency sizes, 970 leaf tasks and above, where the absence of inter-instance links leaves the search space at its most symmetric. The solver’s internal optimality gap was not recorded in this campaign; as a proxy, each feasible makespan can be compared against a proven optimum for the same configuration. This gives a split picture: most feasible runs land within one percent of that optimum, while a minority, again concentrated at the large no-dependency sizes, return schedules materially above it, by around 13% for the largest sparse-template instances and, in two isolated runs at that size, by over 100%. The loose feasible runs and the outright failures thus occupy the same large, dependency-free regime that is the hard case for this problem; at the size and dependency structure of a real stop, a proven optimum is the rule.

The makespan curves reflect the same structure. Without inter-instance dependencies, the instances are scheduled in parallel, so adding more does not lengthen the plan: the makespan stays at 127 days for Template 1 at every solved size, and at 112 days for Template 2 wherever optimality was proven — at the two largest solved sizes (970 and 1,164 tasks), the Template 2 runs returned 127-day schedules without an optimality proof, upper bounds rather than optima (Appendix B.7). With a chain, the instances are serialized and the makespan grows linearly with their number. The practical implication is that the sizes at which the unconstrained case becomes hard, beyond about 1200 tasks, lie well above the leaf-task count of a single programmed stop, and real plans, which carry inter-instance dependencies, fall in the regime where the solver scales without reaching the time limit.

The none-versus-chain contrast spans the two extremes of inter-instance coupling: fully independent instances on one side, a complete serial chain on the other. Real multi-instance plans lie between these, with a limited number of links connecting otherwise independent instances. To characterize this intermediate regime, four instances of the sparser template (388 leaf tasks) were

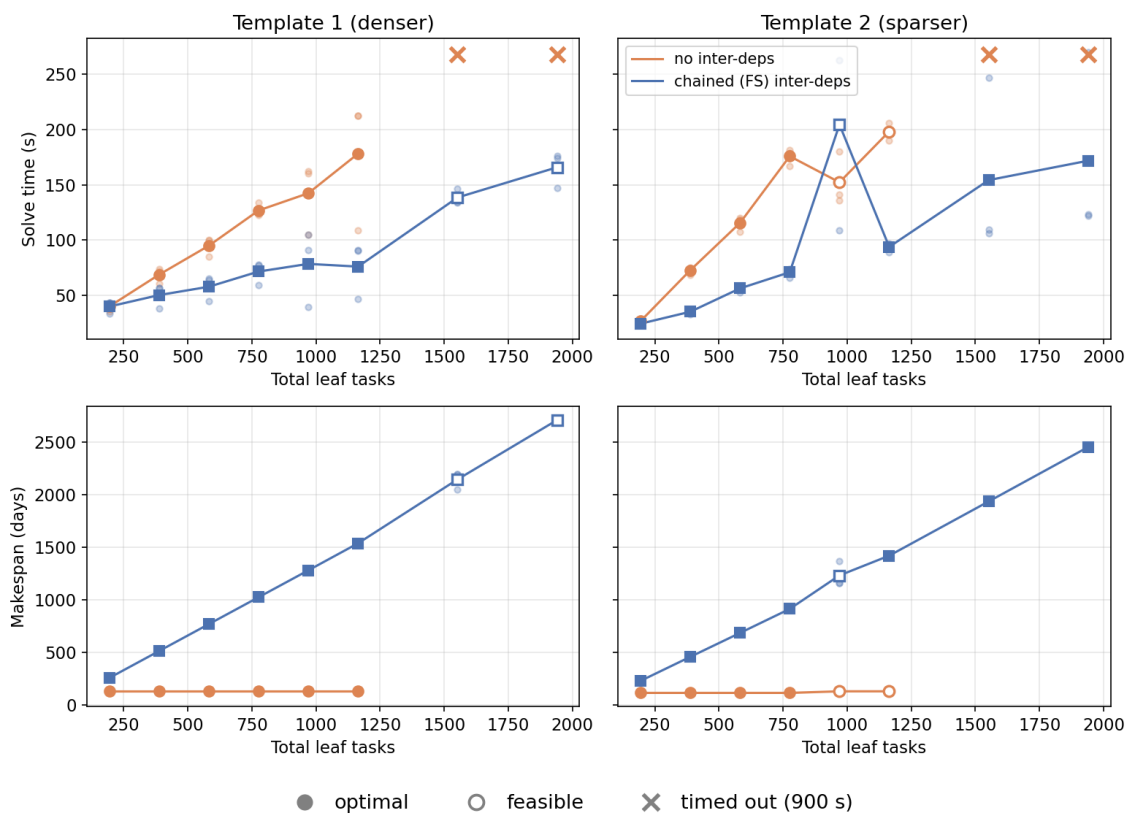


Figure 7.10: Scalability of solve time (top) and makespan (bottom) with problem size, for the denser Template 1 (left) and the sparser Template 2 (right). Each configuration is solved three times; markers distinguish optimal, feasible, and timed-out runs.

solved while the number of inter-instance dependencies was raised from none to sixty-four, each link a finish-to-start dependency between a randomly chosen pair of tasks in different instances; every count was solved three times, with a fresh random choice of links each time. Figure 7.11 reports the mean makespan, and the effect of coupling proves large and early. With no links the four instances run in parallel at the single-instance optimum of 112 days, but only four links already raise the makespan to 257 days, and by sixteen links it saturates at 386 days — near the serial regime, yet still below the 455-day full chain. The rise is steep rather than gradual: a handful of dependencies already exacts most of the serialization penalty. How much a few links cost depends on which tasks they join, the source of the spread at eight links, where the makespan ranged from 257 to 386 days across the three runs; beyond about sixteen links the critical path is fixed regardless of placement. Solve time moved the other way, from 77 seconds at no coupling to around 25 seconds once the links were dense, since each added dependency removes symmetry and shrinks the search. The per-count makespans are tabulated in Table B.13.

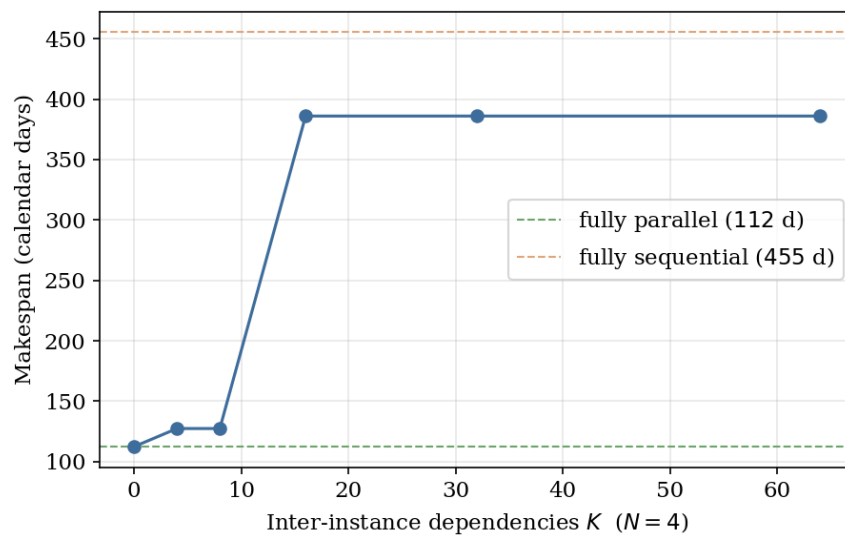


Figure 7.11: Mean makespan over three runs of four jointly scheduled instances of the sparser template as the number of inter-instance dependencies grows. Even a few links push the plan well above the parallel value, and a few tens saturate it near the serial regime, below the full chain; the bar at eight links spans the three runs.

7.4.3 Time Granularity

The time granularity is the size of the time slot into which the solver quantizes the schedule: a finer slot lets tasks start at more closely spaced times but enlarges the model. Figure 7.12 compares 15-, 30-, and 60-minute slots. Coarser slots were uniformly faster at no cost in schedule quality: wherever optimality was proven, the optimal makespan was identical across the three granularities. Finer granularity never improved schedule quality; it only enlarged the model and, on the sparser template at the larger sizes, left optimality unproven or pushed the configuration past the time

limit. The 60-minute granularity is therefore the natural default, and the one the prototype adopts (Section 5.3.4).

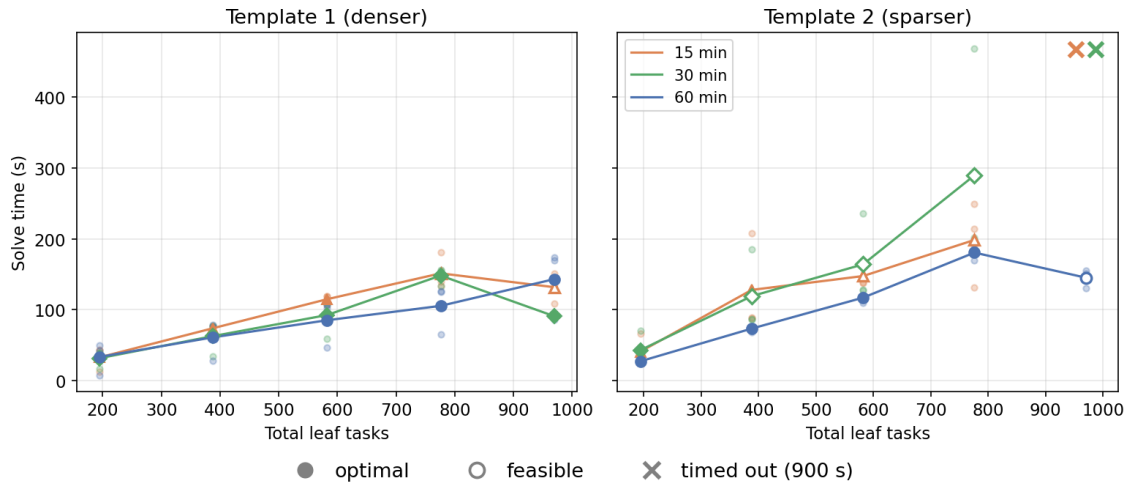


Figure 7.12: Solve time at three time granularities. Finer slots cost solver time with no gain in makespan, which is identical across the three settings wherever optimality was proven.

The choice of a 60-minute slot is reinforced by the granularity of the task durations themselves. Figure 7.13 shows their distribution across the 97 leaf tasks of the template: the shortest is half a working day, two-thirds are one day or less, and every value is a multiple of half a day, with no sub-daily durations at all. The finer 15- and 30-minute settings therefore had no sub-hourly structure to resolve, which is why they bought nothing. A coarser, half-day slot is not used either, because the working-time calendar begins and ends each day at fixed hours (Section 4.5.2), and a 60-minute slot is the coarsest that stays aligned with it. The granularity remains a configurable parameter.

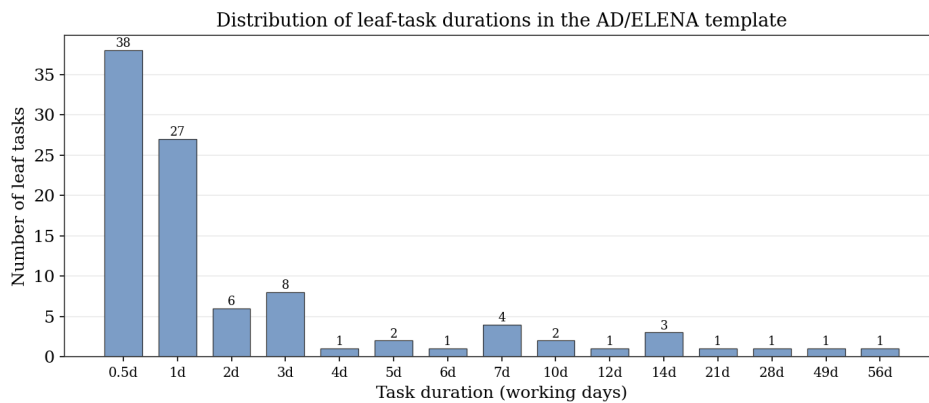


Figure 7.13: Distribution of the 97 leaf-task durations in the benchmark template, in working days. The shortest task is half a working day and every value is a multiple of that unit.

These working-time durations are distinct from the calendar span a task occupies once it is scheduled. Because the model maps working time onto a calendar that excludes weekends, public

holidays, and off-shift hours (Section 4.5.2), a task’s start-to-finish span is generally longer than its duration, and at the level of a whole plan the working-day total and the calendar span it occupies differ substantially. Keeping the duration data coarse is what lets the solver work on a small working-time grid, while the dual working- and calendar-time representation recovers the calendar dates shown to the coordinator.

7.4.4 Warm-Start

The optional Phase 1 warm-start constructs a coarse initial assignment to seed the main search. Figure 7.14 decomposes the wall-clock time of each mode into the solver’s own search time and the overhead of building the hint. A day-level hint did reduce the solver’s search time, but the cost of constructing the hint outweighed that saving, leaving the total wall-clock time comparable to, or worse than, solving without a hint. The week-level hint was the slowest of the three and, more importantly, harmed the schedule itself: the coarse initial assignment trapped the search near a poor region, so that on the denser template the makespan degraded from 127 to as much as 190 days, at the larger sizes, with a similar trap appearing on the sparser template at its largest size. Solving without a warm-start incurred no construction overhead, fell into no such trap, and reached the same makespan as the day-level setting. These results justify disabling the warm-start by default, the configuration described in Section 5.3.5. The remaining experiments used the day-level hint as their base setting, a choice fixed in the benchmark script before these results were known; because the day-level hint reaches the same makespan as solving without one, the makespan conclusions of those experiments are unaffected, and the solve times they report exclude the hint-construction overhead, which is accounted separately in Fig. 7.14.

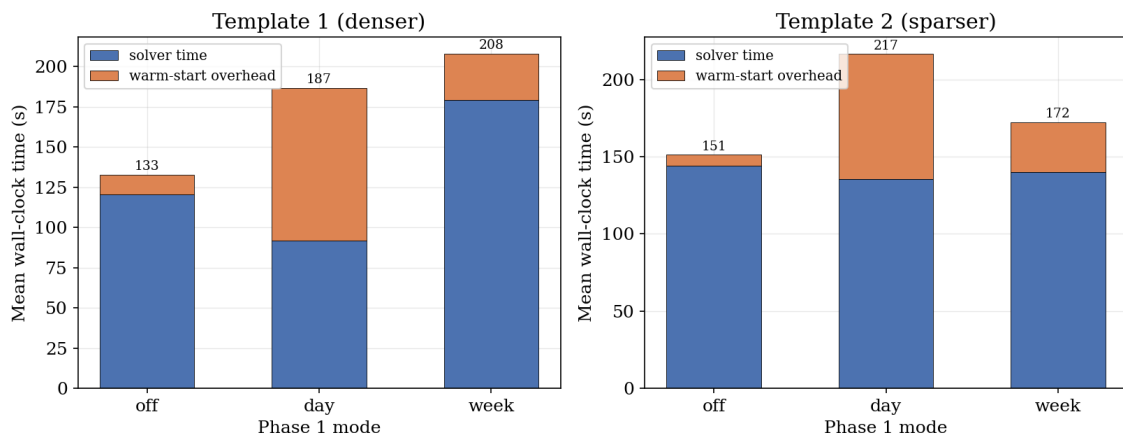


Figure 7.14: Wall-clock time of the warm-start modes, decomposed into the solver’s search time and the hint-construction overhead, for the two templates. The day-level hint’s overhead outweighs the search time it saves; disabling the warm-start is the safe default.

7.4.5 Concurrency Limits

Concurrency limits cap how many tasks competing for the same location or team may run in parallel. Figure 7.15 shows their effect, for two real CERN constraint sets — a location limit and a team limit — on both the makespan and the solve time. The result is the expected one, and it confirms that the limits behave correctly: tighter limits lengthen the schedule, since less work can proceed in parallel, and they also make the model harder to solve. Under the tightest limits the solver did not prove optimality within the budget, so the makespans reported there are upper bounds rather than exact optima; they are also computed over the half-day occupancy buckets of Section 5.3.3, which over-approximate instantaneous concurrency. The effect is monotonic, and once the limit is loosened to three or more the schedule returns to the unconstrained baseline, or to within a day of it. The experiment was also run with the instance chain present: the already-serialized schedule absorbs the limits, and only the tightest limit lengthens it further, so the table in Appendix B.7 reports the independent-instance slice, where the effect is fully visible. The optimizer thus trades parallelism for constraint satisfaction in a predictable way, with the size of the penalty governed by how many tasks contend for the limited resource.

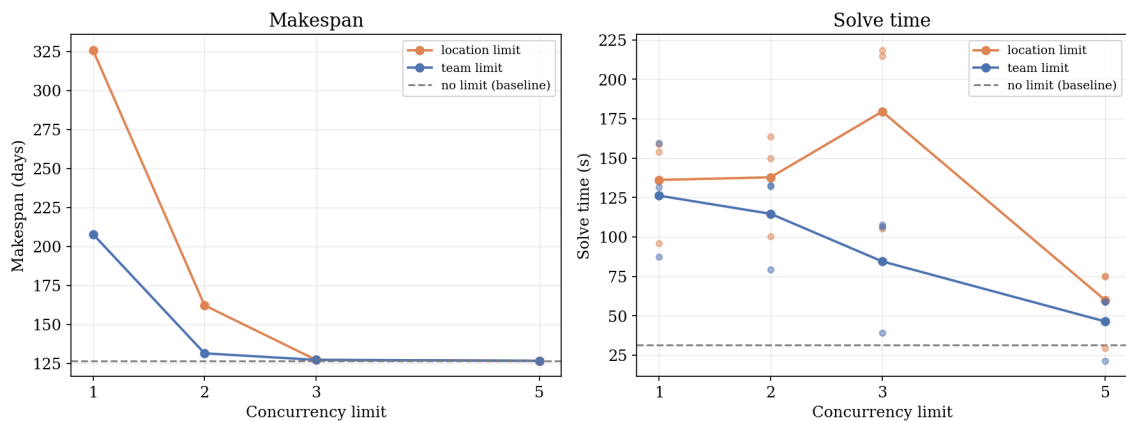


Figure 7.15: Makespan (left) and solve time (right) as a function of the concurrency limit, for a location and a team constraint set, against the unconstrained baseline. Tighter limits lengthen the schedule and increase solve time; both relax to the baseline once the limit reaches three.

The experiment above limits one resource at a time, a single location or a single team. A coordinator may instead need to cap both at once. To examine whether the two limits interact, two instances of the sparser template were solved under a team limit, a location limit, and both together, at maximum-concurrency values of one, two, and three, each configuration solved three times. Figure 7.16 reports the makespans. Each limit alone behaves as before: the team limit, which binds 20 tasks, lengthens the plan from 112 to 197 days at its tightest and relaxes to the baseline by a limit of two, while the location limit, which binds 50 tasks, is the more restrictive of the two and reaches 326 days at a limit of one. Applying both at once does not simply take the larger penalty: at a limit of one the combined makespan of 327 days sits just above the location-only value, and at a limit of two it reaches 170 days, above both the location-only result of 165 and

the team-only result of 112. The two limits therefore compound rather than mask one another, the more binding resource setting the floor and the second adding to it. As in the single-resource case, the tightest combined limits left optimality unproven within the budget, so those makespans are upper bounds, and the penalty vanishes once both limits reach three. The makespans are tabulated in Table B.14.

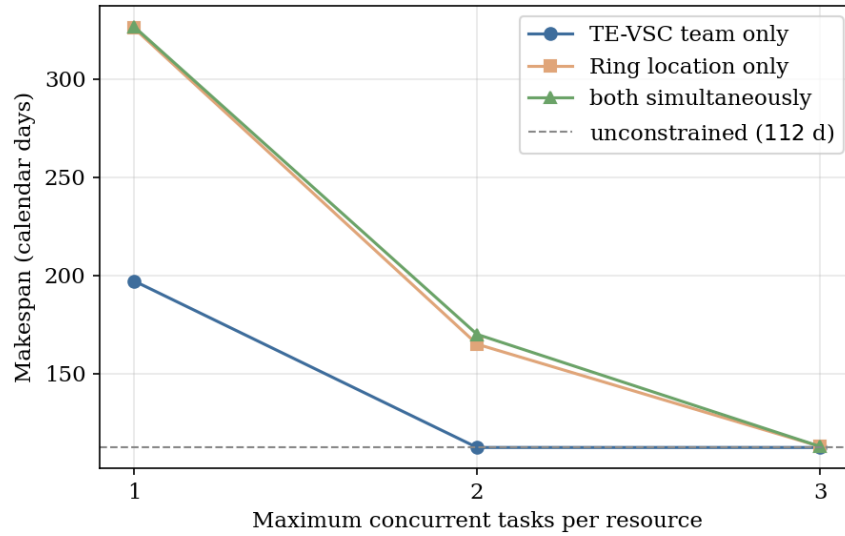


Figure 7.16: Makespan of two jointly scheduled instances of the sparser template under a team limit, a location limit, and both at once, against the unconstrained baseline. The two limits compound: constraining both yields a longer plan than either alone, until both reach a limit of three.

7.4.6 Result Stability

Because the solver explores in parallel without a fixed random seed, repeated solves of the same configuration may follow different search paths (Section 5.3.4); each configuration was therefore solved three times. The makespan proved highly stable across the three runs: for the large majority of configurations the three makespans were identical or differed by less than a tenth of a day. Solve time, by contrast, varied from run to run, with a coefficient of variation around 27% on the denser template and 10% on the sparser one. The few configurations whose makespan did vary were those at the boundary between optimal and merely-feasible: when a run failed to prove optimality within the limit, it returned a schedule above the optimum, usually by a small margin but, at the largest no-dependency sizes, by appreciably more, and this happened more often on the sparser template, whose larger search space makes that boundary harder to cross. The reproducibility of the makespan, even where the solve time fluctuates, confirms that the schedule quality reported here is a property of the model rather than an artifact of any single run.

7.5 Case Study: PS YETS 2025–26

The three tracks evaluate the components in isolation; the case study tests them together, end to end, on a single real programmed stop. The instance is the PS Year-End Technical Stop of 2025–26, a plan of 492 active leaf tasks spanning 70.7 calendar days, drawn directly from the operational scheduling database.

7.5.1 Template Coverage

Six templates were used to reconstruct the plan, each capturing a group of activities that recurs from one YETS to the next. The templates were captured from the stop under study itself, using the import path of Section 5.2.2; what qualifies each group as a template is that it appears, with the same structure, in earlier year-end stops as well. Capturing the six templates took one to two hours in total, most of it spent identifying which task groups recur across stops. The capture carries each group’s hierarchy, durations, dependencies, and constraint dates; no concurrency limits were defined for this plan, so the optimized model is driven by the dependency structure and the working calendars alone. Together the templates cover 364 of the 492 leaf tasks, or 74%. The remaining 26% are project-specific activities, such as individual installations, replacements, and modifications, that do not recur from one stop to the next and therefore do not warrant a template. The optimized model is accordingly built from the six template instances alone; the one-off activities of the remaining 26% — 128 leaf tasks with no recurrence — are not represented in it, and that the makespan still matches the real plan (Section 7.5.2) shows them to be interior to the stop’s envelope rather than part of what defines it. Sixteen cross-template dependencies, three at the template level and thirteen at the task level, were added from the real dependency data to link the instances. Each template is instantiated once in this plan, since this stop repeats no single procedure across many identical units.

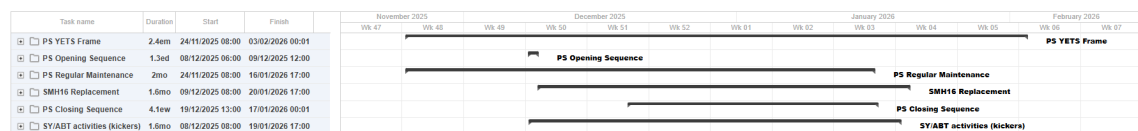
7.5.2 Schedule Reproduction

The assembled schedule was optimized in a single run. The solver returned an optimal solution in 26 seconds. The resulting makespan was 70.7 calendar days, matching the duration of the real plan exactly. Because the templates originate from the plan being reconstructed, this equality is read as a fidelity result rather than an independent discovery: it shows that the abstraction retains the load-bearing structure of the plan — the durations, the dependency chains, and the calendars from which the envelope emerges. The end date of the stop is not imposed on the model: the makespan is a computed outcome, and the solver, which Section 7.5.3 shows moved and reordered tasks extensively, was free to return a shorter or longer schedule had the captured structure allowed one. The optimizer thus reproduces the global envelope of a real year-end stop from reusable templates and a handful of cross-template links, which is the central feasibility claim of this work. Following the framing used throughout, the optimizer is said to reproduce the real plan’s duration,

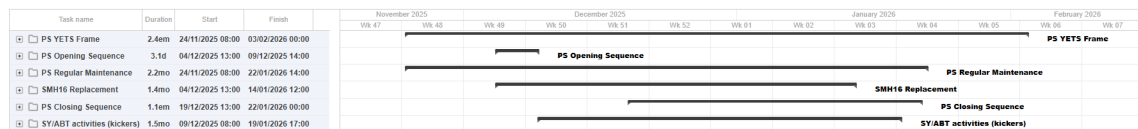
not to better it: the value lies in arriving at the same envelope automatically, from templates, in seconds.

7.5.3 Task-Level Agreement

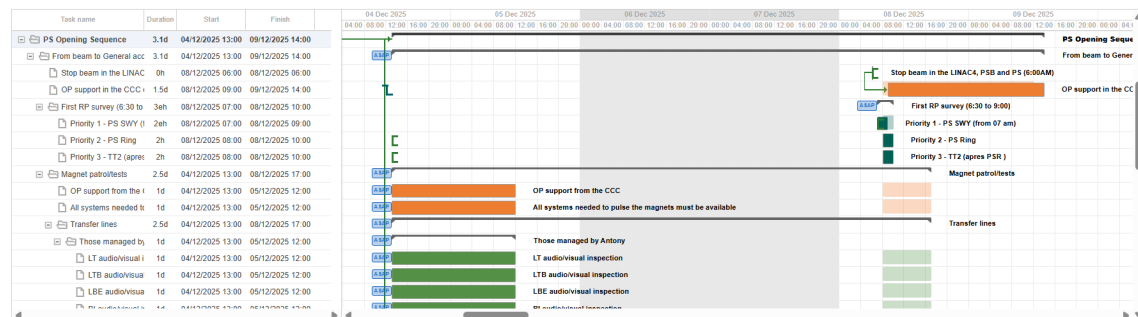
Beyond the overall duration, the optimized start dates were compared task by task with the real plan, measuring each task's deviation from its real start. After optimization, 28% of tasks started on exactly the real date and 59% within one week of it. The agreement is not uniform across the plan, and the pattern is informative. The deterministic parts of the stop — the frame of milestones and the opening and closing sequences — match the real plan closely. The divergence is concentrated in the regular maintenance, which makes up roughly 70% of the schedule: the instance opens on the correct date, but its tasks are reordered internally by the optimizer. Figure 7.17 shows this directly: at the template-group level the optimized plan reproduces the envelope of the real one (panels a and b), while a task-level view of the before-and-after overlay (panel c) exposes the internal reordering, with the optimized bars pulled ahead of the faint ghost bars that mark each task's original position.



(a) Before optimization, at the template-group level.



(b) After optimization: the overall 70.7-day envelope is reproduced.



(c) Task-level detail of the opening sequence: the optimized bars (solid) are pulled ahead of the ghost bars (faint) that mark each task's original position.

Figure 7.17: The PS YETS 2025–26 case study before and after optimization. At the template-group level the optimizer reproduces the overall envelope (a, b); at the task level it reorders tasks within that envelope (c).

This reordering is not a modeling error. Within the envelope fixed by the formal dependencies, many maintenance tasks may be placed in more than one valid order, and the optimizer selects one

such order, guided by its makespan objective and the mild default preference for early starts noted in Section 4.5.3, whereas the real plan reflects the sequence a coordinator would impose from experience, knowledge that is not encoded in the formal constraints. The divergence therefore reflects the implicit planning knowledge that lies outside the model, beyond the optimizer’s own ordering preference.

This is the result that ties the evaluation together. The templates capture roughly three-quarters of a real stop, and the optimizer reproduces its duration exactly; the remaining task-level ordering is where the coordinator’s tacit expertise enters, and it is exactly what the interactive Edit & Review interface (Chapter 4) is designed to let the coordinator supply. The automated tool provides the structure and the global envelope, and the coordinator supplies the implicit-knowledge tail. This division of labor, anticipated by the coordinators in Track B, is validated in a real plan. One concrete way to narrow the residual gap, suggested by a participant, would be to let coordinators attach relative priorities to tasks so that the optimizer’s ordering follows their intent more closely.

7.6 Limitations

Several limitations qualify the results and bound the strength of the claims drawn from them. The usability tasks were guided, scripted walkthroughs, which measure the operability and learnability of individual features rather than a coordinator’s ability to plan a stop unaided; the full task-completion rate should be read in that light. The workflow benefit of Track B is a perceived benefit, elicited by a questionnaire, not a measured reduction in time or effort, since a controlled comparison against the Microsoft Project workflow was not feasible within the session budget. The author served as both the developer of the tool and moderator of the sessions, and the participants were recruited through the project’s CERN supervisor, both of which introduce a risk of demand characteristics and selection bias; the sample of 20 participants, together with a single instance in the case study, is appropriate for a formative evaluation but limits statistical generalization. The sessions also ran to roughly fifty minutes, long enough that fatigue or waning attention could in principle have affected the later tasks; the script mitigated this by placing the most demanding interactions in the middle rather than at the end, though the effect cannot be entirely ruled out.

On the algorithmic side, the optimality reported is optimality with respect to the discretized model — the half-day concurrency buckets and the chosen time granularity — rather than in continuous time, and the objective optimized is the makespan alone, which for a stop with a largely fixed window is a useful first proxy but not necessarily the operational objective. The benchmark scales the problem by replicating two templates of the same accelerator domain, which isolates the effect of constraint density but does not reproduce the heterogeneity of a full long shutdown, and template expressiveness was measured on a single real stop, where it reached 74%. Finally, the optimizer was not compared against a measured baseline — a manual scheduler, a heuristic, or a public benchmark of resource-constrained scheduling with calendars — so Track C characterizes the approach rather than ranking it against alternatives.

7.7 Summary

This chapter reported the outcomes of the evaluation across its three tracks and the case study. The study involved 20 CERN members — 14 planning coordinators and 6 in non-planning roles — spanning a wide range of age and Microsoft Project experience, working on a facility that most did not know in advance.

In the usability track, every participant completed the eight scripted tasks, with the time, errors, and assists concentrating on the two richest interactions, editing a template and creating a schedule by instantiation, while the linear open and export flows were friction-free. Participants rated every task both easy and useful, and the System Usability Scale returned a mean of 81.5 — a grade-A score that did not depend on planning background, indicating that the interface is approachable to expert and occasional users alike. The open-ended feedback was strongly positive about the templates and the optimizer and converged on a few concrete refinements, mainly more transparency around the optimization step and schedule-level undo and redo.

The workflow questionnaire found the coordinators confident in the output and willing to adopt the tool, with one reservation, the optimizer’s handling of implicit knowledge. The algorithmic benchmark showed that the optimizer scales well beyond the size of a single stop when inter-instance dependencies are present, that a coarse time granularity is both faster and lossless, that the warm-start is best disabled, and that concurrency limits lengthen the schedule in proportion to resource contention, while the makespan itself is reproducible across runs. The coordinators’ reservation is precisely echoed by the case study, which reconstructed a real year-end stop from six templates covering 74% of its tasks and reproduced its makespan exactly. These results are taken up in the conclusions of Chapter 8.

Chapter 8

Conclusions

This chapter concludes the dissertation. It first revisits the problem and synthesizes what each part of the work established, together with the limitations that bound it; it then returns to the research questions and objectives in the light of the evidence gathered in Chapter 7 and draws together the contributions; and it finally sets out the directions the work opens for future development.

8.1 Context and Synthesis

The planning of programmed stops at CERN, the recurring technical stops during which the accelerators are maintained and upgraded, is a substantial coordination problem: hundreds of interdependent interventions must be sequenced under tight temporal, resource, and safety constraints, and the quality of that sequencing bears directly on how long the machines remain off. This planning is still carried out largely by hand in Microsoft Project, a tool that records a schedule but offers little help in reasoning about how its many constraints interact, and rebuilds each recurring plan from scratch rather than from a reusable abstraction.

The gap addressed by this dissertation is therefore not the absence of a scheduler but the absence of an integrated approach that lets a recurring plan be authored once, optimized under realistic constraints, and reviewed by the coordinator who owns it. Existing tools support either the visual inspection of a schedule or its optimization, but rarely both over one reusable abstraction, which leaves coordinators to translate by hand between the way they read a plan and the way an optimizer would compute one.

The work approached this gap in stages, and each chapter established a piece of the result. The review of the visualization and scheduling literature (Chapter 3) found the two halves of the problem, the visual and the algorithmic, to be well studied in isolation but rarely integrated over a reusable plan, which framed the contribution as one of integration rather than of a new algorithm. The solution chapter (Chapter 4) defined the template abstraction at the center of that integration, together with the conceptual architecture and the design decisions around it, and the implementation chapter (Chapter 5) realized them as an extension of the CERN Scheduling Tools platform, with a constraint-programming optimizer over a dual working- and calendar-time

representation. The evaluation (Chapters 6 and 7) then exercised the result along three tracks and a real case study, and it is that evidence the rest of this chapter draws on.

That evidence is qualified in several ways, set out in full in Section 7.6 and summarized here as the frame within which the conclusions below should be read. The usability study was formative, built on guided tasks rather than unaided planning; the workflow comparison captured a perceived rather than a measured benefit; the author was both the developer of the tool and the moderator of the sessions; and the case study, like the benchmark, rests on a single accelerator domain and a single real stop. The findings that follow are therefore best read as characterizations of what the approach makes possible, not as generalizable measurements of how far it improves on current practice.

8.2 Research Questions and Objectives

This section returns to the two research questions and the four objectives in the light of the evidence summarized above, answering each in turn, and then draws together the contributions of the work.

RQ1. The first research question asked what visual representation techniques and visual-language principles can be employed to model, display, and interact with complex scheduling constraints in task and project planning systems. The prototype answers this question constructively, through a small and coherent set of techniques layered on an interactive timeline: a semantically transparent notation for dependencies and scheduling constraints, a concurrency heatmap for resource contention, and a before-and-after overlay with reason-annotated change lists. The usability evaluation confirms that they work in practice: coordinators and non-planning members alike read, edited, and reasoned about constraint-rich schedules with high success, low perceived difficulty, and a System Usability Scale score of 81.5 that did not appear to depend on prior planning experience.

RQ2. The second research question asked how constraint-rich project scheduling in this context can be modeled and solved, and what trade-offs between schedule quality and computational cost the resulting approach exhibits. The problem was modeled as a constraint-programming formulation over a dual working- and calendar-time representation and solved with CP-SAT. The algorithmic benchmark characterized the trade-offs: solve time grows roughly linearly with problem size and stays well within a practical budget for the size of a single stop, provided inter-instance dependencies are present; a coarse time granularity is both faster and lossless; the warm-start is best disabled; and concurrency limits lengthen the schedule in proportion to resource contention. The case study then closed the loop on a real plan, where the formulation reproduced the duration of the real year-end stop exactly, in seconds.

Objectives. Within the scope of this evaluation — a formative study with guided tasks and a single-stop case study — the four objectives are met. The template abstraction proved expressive enough for coordinators to extract, edit, and reuse after only a short introduction, and it covered 74% of a real stop (O1); the optimizer was characterized across problem sizes and parameters and returned interpretable, optimal schedules on real-template instances (O2); the prototype was found highly usable, with no dependence on planning background (O3); and the seven coordinators who completed the workflow comparison judged it an improvement over their current practice, all of them willing to consider adopting it (O4), although that benefit remains a perceived rather than a measured one. Three things lie beyond this scope and are not claimed here: a measured baseline comparison for the optimizer, a field study of real adoption, and generalization beyond this single CERN setting; these are taken up as future work in Section 8.3.

Beyond answering these questions, the work makes two main contributions. The first, and its conceptual core, is an integrated, template-centric workflow that closes the loop from authoring a reusable plan, through optimization, to a reviewed export, a combination that existing planning tools, which support either visual inspection or optimization but rarely both over a reusable abstraction, do not provide. The second is a human-in-the-loop review interface whose necessity the case study underscores: the optimizer supplies the global structure and the temporal envelope, while the coordinator supplies the implicit, experiential knowledge that lies outside any formal model. This interface is paired with an explainable infeasibility diagnosis that turns a solver rejection into typed, fixable violations, so that the coordinator stays in control even when the model has no solution. The optimization itself is expressed with only standard constraints over a dual working- and calendar-time representation, which keeps it independent of any specialized calendar algorithm. The central claim of this dissertation is therefore not a new scheduling algorithm but an integrated, template-centric approach: a reusable template abstraction and the workflow built around it.

8.3 Future Work

The evaluation and the case study point to several directions for future work, grouped below by theme.

Extending the optimization model. The formulation optimizes a single objective over the temporal and resource dimensions; it could be extended to

- balance the makespan against competing objectives such as resource leveling, cost, or radiation exposure, in a multi-objective formulation;
- support what-if analysis, running several optimizations under different parameters and comparing the outcomes side by side;
- let the coordinator attach relative priorities to tasks, steering the optimizer’s ordering toward their intent;

- fold in the safety, access, and spatial dimensions that separate CERN processes and tools currently handle in isolation, for instance by deriving no-overlap constraints between interventions whose locations are adjacent or share access paths, from the data those tools already hold;
- scale to the far larger and more heterogeneous plans of Long Shutdowns; and
- explore a multi-agent formulation, in which several coordinators plan different facilities in parallel and negotiate shared resources and access windows rather than solving one centralized model, matching the distributed way large shutdowns are coordinated in practice.

Learning from coordinators. The case study located the residual gap between the optimized and the real schedule in the implicit, experiential knowledge that coordinators apply but rarely formalize. A natural next step is to capture this knowledge by

- mining recurring patterns from coordinators' manual overrides (for instance, two tasks that are never scheduled concurrently in practice) and proposing them as explicit constraints; and
- suggesting constraints and durations for a new plan from a corpus of past stops, following a case-based reasoning approach that reuses concrete precedents rather than a purely data-driven one.

Refining the interface. The evaluation surfaced a number of concrete refinements, the most frequently raised being

- greater transparency around the optimization step, including a side-by-side before-and-after view and an indicator of the solver's progress while it runs;
- undo and redo at the level of the schedule, and clearer signposting of the accept-and-revert workflow;
- a broader overview of the Gantt chart, with zoom and a critical-path view;
- richer editing of the task hierarchy, by reordering tasks and moving them to a different parent;
- suggesting, when the author marks one occurrence of a placeholder, the other places where the same value appears, so that no occurrence is overlooked;
- propagating an edit of a template to the instances already created from it, for changes that become known only after a first version of the schedule exists;
- export to formats beyond Microsoft Project XML, such as native project files, spreadsheets, and PDF;

- better discoverability of features on first use, through in-app help and an FAQ section; and
- multi-user collaboration with an audit trail of who changed what, when, and why.

Broadening the evaluation. Finally, several directions would strengthen the evidence and carry the tool toward operational use:

- a measured comparison against a baseline (the manual Microsoft Project workflow, a scheduling heuristic, or a public benchmark of resource-constrained scheduling with calendars), which would also improve reproducibility;
- targeted usability studies of the features the present evaluation could not isolate, among them the constraint notation read without its legend and tooltips, the named placeholders, the conflict resolver, and the template lifecycle;
- building out a complete, validated library of templates for each facility, which the coordinators identified as the principal condition for replacing their current workflow; and
- a field study of real adoption, following the coordinators' stated willingness to use the tool in production.

Taken together, these directions would deepen the optimization, let the system learn the tacit knowledge that the case study found to lie outside the model, and carry the tool from a validated prototype toward everyday operational use.

In conclusion, this dissertation has demonstrated that the planning of programmed stops at CERN can be supported end to end by a single, template-centric workflow, in which a recurring plan is authored once, optimized under realistic constraints, and reviewed and exported with the coordinator in the loop. The constraint-programming formulation proved both expressive and tractable at the scale of a real stop; the usability evaluation showed the interface to be approachable to expert and occasional users alike; and the case study confirmed, on a real year-end stop, that the approach reproduces a human plan's envelope automatically while leaving the coordinator's implicit expertise where it belongs. The contributions and the future work set out here provide a foundation for that workflow to mature and, more broadly, for the integration of visual analytics and optimization in constraint-rich planning beyond CERN.

References

- Aigner, W., Miksch, S., Schumann, H., & Tominski, C. (2011). *Visualization of time-oriented data* [ISBN: 978-0-85729-078-6]. Springer.
- Artigues, C., Hartmann, S., & Vanhoucke, M. (2026). Fifty years of research on resource-constrained project scheduling explored from different perspectives [DOI: 10.1016/j.ejor.2025.03.024]. *European Journal of Operational Research*, 328(2), 367–389.
- Bangor, A., Kortum, P. T., & Miller, J. T. (2008). An empirical evaluation of the system usability scale [DOI: 10.1080/10447310802205776]. *International Journal of Human–Computer Interaction*, 24(6), 574–594.
- Behrens, J. K., Lange, R., & Mansouri, M. (2019). A constraint programming approach to simultaneous task allocation and motion scheduling for industrial dual-arm manipulation tasks [DOI: 10.1109/ICRA.2019.8794022]. *Proceedings of the 2019 International Conference on Robotics and Automation (ICRA)*, May 20–24, 2019, Montreal, Canada, 8705–8711.
- Borreguero Sanchidrián, T., Portoleau, T., Artigues, C., García Sánchez, A., Ortega Mier, M., & Lopez, P. (2024). Large neighborhood search for an aeronautical assembly line time-constrained scheduling problem with multiple modes and a resource leveling objective [DOI: 10.1007/s10479-023-05629-3]. *Annals of Operations Research*, 338(1), 13–40.
- Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology [DOI: 10.1191/1478088706qp063oa]. *Qualitative Research in Psychology*, 3(2), 77–101.
- Brooke, J. (1996). SUS: A quick and dirty usability scale [ISBN: 978-0-7484-0460-5]. In P. W. Jordan, B. Thomas, B. A. Weerdmeester, & I. L. McClelland (Eds.), *Usability evaluation in industry* (pp. 189–194). Taylor & Francis.
- Brucker, P. (2007). *Scheduling algorithms* (5th edition) [ISBN: 978-3-540-69515-8]. Springer.
- Brucker, P., Drexl, A., Möhring, R., Neumann, K., & Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models, and methods [DOI: 10.1016/S0377-2217(98)00204-5]. *European Journal of Operational Research*, 112(1), 3–41.
- Burke, E. K., De Causmaecker, P., Vanden Berghe, G., & Van Landeghem, H. (2004). The state of the art of nurse rostering [DOI: 10.1023/B:JOSH.0000046076.75950.0b]. *Journal of Scheduling*, 7(6), 441–499.
- Caprace, J.-D., Petcu, C., Velarde, M., & Rigo, P. (2013). Optimization of shipyard space allocation and scheduling using a heuristic algorithm [DOI: 10.1007/s00773-013-0217-2]. *Journal of Marine Science and Technology*, 18(3), 404–417.
- Ceschia, S., Rosati, R. M., Schaerf, A., Smet, P., Vanden Berghe, G., & Zanazzo, E. (2025). The integrated healthcare timetabling competition 2024 [DOI: 10.1016/j.ordal.2025.200481]. *Operations Research, Data Analytics and Logistics*, 45, 200481.
- Chang, H., Yu, W., & Cheng, S. (2017). A risk-based critical path scheduling method (II): A visual approach system using BIM [DOI: 10.22260/isarc2017/0073]. *Proceedings of the 34th International Symposium on Automation and Robotics in Construction (ISARC 2017)*, July 28–July 1, 2017, Taipei, Taiwan, 527–535.

- Chen, A. H. .-, Liang, Y. .-, & Padilla, J. D. (2017). Using discrete differential evolution and entropy to solve the MRCPSP [DOI: 10.1109/CEC.2017.7969600]. *Proceedings of the 2017 IEEE Congress on Evolutionary Computation (CEC), June 5–8, 2017, San Sebastian, Spain*, 2437–2442.
- Cockburn, A., Karlson, A., & Bederson, B. B. (2009). A review of overview+detail, zooming, and focus+context interfaces [DOI: 10.1145/1456650.1456652]. *ACM Computing Surveys*, 41(1), 2:1–2:31.
- Das, P. P., & Acharyya, S. (2011). Meta-heuristic approaches for solving resource constrained project scheduling problem: A comparative study [DOI: 10.1109/CSAE.2011.5952511]. *Proceedings of the 2011 IEEE International Conference on Computer Science and Automation Engineering, June 10–12, 2011, Shanghai, China*, 474–478.
- Demeulemeester, E. L., & Herroelen, W. S. (2002). *Project scheduling: A research handbook* [ISBN: 978-1-4020-7051-8]. Springer.
- Desai, N. V., Yadav, N. B., & Malaviya, N. N. (2023). Increasing the potential application of microsoft project and primavera p6 for project management: A comparative analysis of the residential project [DOI: 10.1016/j.matpr.2022.11.485]. *Materials Today: Proceedings*, 77, 794–804.
- DHTMLX. (2021). *DHTMLX gantt: JavaScript gantt chart component* [Version 7.1.8]. Retrieved June 2, 2026, from <https://docs.dhtmlx.com/gantt/>
- Dorndorf, U. (2012). *Project scheduling with time windows: From theory to applications* [ISBN: 978-1-4615-1613-2]. Springer Science & Business Media.
- Eberhart, R., & Kennedy, J. (1995). A new optimizer using particle swarm theory [DOI: 10.1109/MHS.1995.494215]. *Proceedings of the 6th International Symposium on Micro Machine and Human Science (MHS'95), October 4–6, 1995, Nagoya, Japan*, 39–43.
- Ernst, A. T., Jiang, H., Krishnamoorthy, M., & Sier, D. (2004). Staff scheduling and rostering: A review of applications, methods and models [DOI: 10.1016/S0377-2217(03)00095-X]. *European Journal of Operational Research*, 153(1), 3–27.
- Ertek, G., Choi, B. .-, Chi, X., Yang, D., & Yong, O. B. (2015). Graph-based analysis of resource dependencies in project networks [DOI: 10.1109/BigData.2015.7363999]. *Proceedings of the 2015 IEEE International Conference on Big Data (Big Data), October 29–November 1, 2015, Santa Clara, CA, USA*, 2143–2149.
- Folco, P., Sahli, A., Belmokhtar-Berraf, S., Bouillaut, L., Fayemi, P. .-, & Turgis, F. (2023). An ILP for scheduling rolling stock maintenance activities under operational and logistical constraints [DOI: 10.1109/CoDIT58514.2023.10284499]. *Proceedings of the 9th International Conference on Control, Decision and Information Technologies (CoDIT), 2023*, 1313–1318.
- Franck, B., Neumann, K., & Schwindt, C. (2001). Project scheduling with calendars [DOI: 10.1007/PL00013355]. *OR Spektrum*, 23(3), 325–334.
- Ghoddousi, P., Eshtehadian, E., Jooybanpour, S., & Javanmardi, A. (2013). Multi-mode resource-constrained discrete time-cost-resource optimization in project scheduling using non-dominated sorting genetic algorithm [DOI: 10.1016/j.autcon.2012.11.014]. *Automation in Construction*, 30, 216–227.
- Graham, R. L., Lawler, E. L., Lenstra, J. K., & Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey [DOI: 10.1016/S0167-5060(08)70356-X]. *Annals of Discrete Mathematics*, 5, 287–326.
- Hartmann, S. (2021). Optimization models and solution techniques [DOI: 10.1007/978-3-030-81123-5_2]. In *Application of mathematics and optimization in construction project management* (pp. 25–50). Springer International Publishing.

- Hartmann, S., & Briskorn, D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem [DOI: 10.1016/j.ejor.2009.11.005]. *European Journal of Operational Research*, 207(1), 1–14.
- Hegazy, T., Saad, D., & Mostafa, K. (2020). Enhanced repetitive-scheduling computation and visualization [DOI: 10.1061/(ASCE)CO.1943-7862.0001911]. *Journal of Construction Engineering and Management*, 146(10), 04020118.
- Hu, Z., Wang, F., & Tang, Y. (2025). Cost optimization of repetitive project scheduling through a constraint programming-based relax-and-solve algorithm [DOI: 10.1016/j.autcon.2025.106272]. *Automation in Construction*, 176, 106272.
- Iles, J. (2019). *MPXJ: Read and write project plans* [Version 7.8.1]. Retrieved June 2, 2026, from <https://www.mpxj.org/>
- Ivson, P., Nascimento, D., Celes, W., & Barbosa, S. D. (2018). CasCADE: A novel 4D visualization system for virtual construction planning [DOI: 10.1109/TVCG.2017.2745105]. *IEEE Transactions on Visualization and Computer Graphics*, 24(1), 687–697.
- Jamieson, S. (2004). Likert scales: How to (ab) use them? [DOI: 10.1111/j.1365-2929.2004.02012.x]. *Medical Education*, 38(12), 1217–1218.
- Jamnuch, R., & Vatanawood, W. (2019). Transforming activity network diagram with timed petri nets [DOI: 10.1109/ICTS.2019.8850996]. *Proceedings of the 12th International Conference on Information & Communication Technology and System (ICTS)*, July 18–19, 2019, Surabaya, Indonesia, 125–129.
- Jo, J., Huh, J. F., Park, J., Kim, B., & Seo, J. (2014). LiveGantt: Interactively visualizing a large manufacturing schedule [DOI: 10.1109/TVCG.2014.2346454]. *IEEE Transactions on Visualization and Computer Graphics*, 20(12), 2329–2338.
- Jung, S., Cho, J. B., Yang, K., Kim, D., & Kim, W. (2025). Efficient task scheduling using constrains programming [DOI: 10.1109/ICCE63647.2025.10930039]. *Proceedings of the 2025 IEEE International Conference on Consumer Electronics (ICCE)*, January 2025, Las Vegas, NV, USA, 1–5.
- Karam, A., & Lazarova-Molnar, S. (2013). Recent trends in solving the deterministic resource constrained project scheduling problem [DOI: 10.1109/Innovations.2013.6544405]. *Proceedings of the 9th International Conference on Innovations in Information Technology (Innovations'13)*, March 17–19, 2013, Al-Ain, United Arab Emirates, 124–129.
- Keim, D. A., Mansmann, F., Schneidewind, J., Thomas, J., & Ziegler, H. (2008). Visual analytics: Scope and challenges [DOI: 10.1007/978-3-540-71080-6_6]. In *Visual data mining* (pp. 76–90). Springer Berlin Heidelberg.
- Kelley, J. E., & Walker, M. R. (1959). Critical-path planning and scheduling [DOI: 10.1145/1460299.1460318]. *Proceedings of the Eastern Joint IRE-AIEE-ACM Computer Conference, December 1–3, 1959, Boston, MA, USA*, 160–173.
- Kendall, G., Knust, S., Ribeiro, C. C., & Urrutia, S. (2010). Scheduling in sports: An annotated bibliography [DOI: 10.1016/j.cor.2009.05.013]. *Computers & Operations Research*, 37(1), 1–19.
- Kim, J., Zhang, H., André, P., Chilton, L., MacKay, W., Lafon, M., Miller, R., & Dow, S. (2013). Cobi: A community-informed conference scheduling tool [DOI: 10.1145/2501988.2502034]. *Proceedings of the 26th annual ACM symposium on User interface software and technology (UIST '13)*, October 8–11, 2013, St. Andrews, UK, 173–182.
- Klein, N., Gnägi, M., & Trautmann, N. (2024). Mixed-integer linear programming for project scheduling under various resource constraints [DOI: 10.1016/j.ejor.2024.06.036]. *European Journal of Operational Research*, 319(1), 79–88.

- Kolisch, R., & Sprecher, A. (1997). PSPLIB - a project scheduling problem library [DOI: 10.1016/S0377-2217(96)00170-1]. *European Journal of Operational Research*, 96(1), 205–216.
- Kreter, S., Schutt, A., & Stuckey, P. J. (2015). Modeling and solving project scheduling with calendars [DOI: 10.1007/978-3-319-23219-5_19]. *Principles and Practice of Constraint Programming (CP 2015)*, 9255, 262–278.
- Lerch, D., & Trautmann, N. (2019). A lazy-constraints approach to resource-constrained project scheduling [DOI: 10.1109/IEEM44572.2019.8978524]. *Proceedings of the 2019 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), December 15–18, 2019, Macao, China*, 144–148.
- Leung, J., Kelly, L., & Anderson, J. H. (2004). *Handbook of scheduling: Algorithms, models, and performance analysis* [ISBN: 978-1-58488-397-5]. CRC Press, Inc.
- Lewis, J. R., & Sauro, J. (2009). The factor structure of the system usability scale [DOI: 10.1007/978-3-642-02806-9_12]. *Human Centered Design: First International Conference (HCD 2009), Held as Part of HCI International 2009*, 94–103.
- Liu, J., Chen, J., Tian, J., & Li, Y. (2025). An efficient random key-based genetic algorithm for solving multimode resource-constrained project scheduling problem [DOI: 10.1061/JCEMD4.COENG-16722]. *Journal of Construction Engineering and Management*, 151(12), 04025193.
- Luz, S., Masoodian, M., McKenzie, D., & Broeck, W. (2009). Chronos: A tool for interactive scheduling and visualisation of task hierarchies [DOI: 10.1109/IV.2009.88]. *Proceedings of the 13th International Conference Information Visualisation (IV'09), July 2009, Barcelona, Spain*, 241–246.
- Ma, N., Li, H., Liu, H., & Yang, L. (2023). Optimal refinery crude oil scheduling based on proximal policy optimization [DOI: 10.1109/ICCC59590.2023.10507286]. *Proceedings of the 9th International Conference on Computer and Communications (ICCC), 2023*, 2449–2453.
- Ma, Z., Yu, S., Yue, P., Cui, D., & Dai, X. (2025). Optimal traffic and task scheduling for track laying in high-speed rail construction [DOI: 10.23919/CCC64809.2025.11178377]. *Proceedings of the 44th Chinese Control Conference (CCC), 2025*, 1945–1950.
- Malcolm, D. G., Roseboom, J. H., Clark, C. E., & Fazar, W. (1959). Application of a technique for research and development program evaluation [DOI: 10.1287/opre.7.5.646]. *Operations Research*, 7(5), 646–669.
- Maniezzo, V., Stützle, T., & Voß, S. (2021). *Matheuristics* [ISBN: 978-3-030-70276-2]. Springer.
- Matos, J. F. C. d. S. d. (2025). *Spatiotemporal task planning visualization in evolving large 3D models using temporal focus + context* [Master's thesis]. Faculty of Engineering, University of Porto (FEUP).
- Moody, D. (2009). The physics of notations: Toward a scientific basis for constructing visual notations in software engineering [DOI: 10.1109/TSE.2009.67]. *IEEE Transactions on Software Engineering*, 35(6), 756–779.
- Morin-Pepin, S., & Francis, A. (2018). Calculation of the construction sites' occupancy rate using chonographic modeling. *Proceedings of the 35th International Symposium on Automation and Robotics in Construction (ISARC 2018)*, 653–659.
- Munzner, T. (2014). *Visualization analysis and design* [ISBN: 978-1-4665-0891-0]. A K Peters/CRC Press.
- Ning, X., Qi, J., Wu, C., & Wang, W. (2018). A tri-objective ant colony optimization based model for planning safe construction site layout [DOI: 10.1016/j.autcon.2018.01.007]. *Automation in Construction*, 89, 1–12.

- Nisar, S., & Suzuki, K. (2015). Critical activity analysis in precedence diagram method scheduling network [DOI: 10.1007/978-3-319-27680-9_14]. *Proceedings of the 11th International Conference on Intelligent Information Processing (IIP 2015)*, 2015, 232–247.
- Perron, L., & Didier, F. (2025). *CP-SAT solver* [Google OR-Tools, version 9.14]. Retrieved June 2, 2026, from https://developers.google.com/optimization/cp/cp_solver/
- Perrot, A.-L., et al. (2022). The second long shutdown of the LHC and its injectors: Feedback from the accelerator coordination and engineering group [DOI: 10.18429/JACoW-IPAC2022-WEPOTK010]. *Proceedings of the 13th International Particle Accelerator Conference (IPAC'22)*, 2052–2055.
- Pinedo, M. L. (2022). *Scheduling: Theory, algorithms, and systems* (6th edition) [ISBN: 978-3-031-05920-9]. Springer International Publishing.
- Project Management Institute. (2011). *Practice standard for scheduling* (2nd edition) [ISBN: 978-1-935589-24-2].
- Rahman, M. M., Bobadilla, L., Mostafavi, A., Carmenate, T., & Zanlongo, S. A. (2018). An automated methodology for worker path generation and safety assessment in construction projects [DOI: 10.1109/TASE.2016.2628898]. *IEEE Transactions on Automation Science and Engineering*, 15(2), 479–491.
- Ramin, D., Leo, E., Nicolosi, L., Spinelli, S., & Brusafferri, A. (2020). A unified PSO-based method for multi-hoist scheduling in advanced galvanic plants [DOI: 10.1109/CoDIT49905.2020.9263919]. *Proceedings of the 7th International Conference on Control, Decision and Information Technologies (CoDIT)*, June 29–July 2, 2020, Prague, Czech Republic, 656–661.
- Roberts, J. C. (2007). State of the art: Coordinated & multiple views in exploratory visualization [DOI: 10.1109/CMV.2007.20]. *Proceedings of the 5th International Conference on Coordinated and Multiple Views in Exploratory Visualization (CMV 2007)*, July 2–3, 2007, Zurich, Switzerland, 61–71.
- Rossi, F., Van Beek, P., & Walsh, T. (2008). Constraint programming [DOI: 10.1016/S1574-6526(07)03004-0]. In F. van Harmelen, V. Lifschitz, & B. Porter (Eds.), *Handbook of knowledge representation* (pp. 181–211, Vol. 3). Elsevier.
- Sakin, S. A., & Isaacs, K. E. (2024). A literature-based visualization task taxonomy for gantt charts [DOI: 10.1109/VIS55277.2024.00055]. *Proceedings of the 2024 IEEE Visualization and Visual Analytics (VIS)*, October 13–18, 2024, Florida, USA, 236–240.
- San Cristóbal Mateo, J. R. (2019). An integer linear programming model including time, cost, quality, and safety [DOI: 10.1109/ACCESS.2019.2953185]. *IEEE Access*, 7, 168307–168315.
- Sánchez, M. G., Gil, A. F., & Castro, C. (2019). Integrating a SMT solver based local search in ant colony optimization for solving RCMPSP [DOI: 10.1109/LA-CCI47412.2019.9036765]. *Proceedings of the 2019 IEEE Latin American Conference on Computational Intelligence (LA-CCI)*, November 11–15, 2019, Guayaquil, Ecuador, 1–6.
- Schaerf, A. (1999). A survey of automated timetabling [DOI: 10.1023/A:1006576209967]. *Artificial Intelligence Review*, 13(2), 87–127.
- Schneider, T., & Aigner, W. (2011). A-plan: Integrating interactive visualization with automated planning for cooperative resource scheduling [DOI: 10.1145/2024288.2024341]. *Proceedings of the 11th International Conference on Knowledge Management and Knowledge Technologies (i-KNOW '11)*, 1–8.

- Semenov, V., Anichkin, A., Morozov, S., Tarlapan, O., & Zolotov, V. (2013). Visual planning and scheduling of industrial projects with spatial factors [DOI: 10.3233/978-1-61499-302-5-343]. *Proceedings of the 20th ISPE International Conference on Concurrent Engineering (CE 2013), September 2013, Madison, WI, USA*, 343–352.
- Shneiderman, B. (1996). The eyes have it: A task by data type taxonomy for information visualizations [DOI: 10.1109/VL.1996.545307]. *Proceedings of the 1996 IEEE Symposium on Visual Languages (VL '96)*, 336–343.
- Soltan, S., & Ashrafi, M. (2025). A new optimization model for multi-project scheduling considering dynamic resource allocation [DOI: 10.1007/s00500-025-10891-7]. *Soft Computing*, 29(15-16), 5143–5158.
- Sun, D., Huang, R., Chen, Y., Wang, Y., Zeng, J., Yuan, M., Pong, T. .-, & Qu, H. (2020). PlanningVis: A visual analytics approach to production planning in smart factories [DOI: 10.1109/TVCG.2019.2934275]. *IEEE Transactions on Visualization and Computer Graphics*, 26(1), 622–632.
- Tarjan, R. (1972). Depth-first search and linear graph algorithms [DOI: 10.1137/0201010]. *SIAM Journal on Computing*, 1(2), 146–160.
- Thomas, J. J., Khader, A. T., Belaton, B., & Christy, E. (2010). Visual interface tools to solve real-world examination timetabling problem [DOI: 10.1109/CGIV.2010.36]. *Proceedings of the 7th International Conference on Computer Graphics, Imaging and Visualization (CGIV 2010), August 7–10, 2010, Sydney, Australia*, 167–172.
- Tominski, C., Gladisch, S., Kister, U., Dachsel, R., & Schumann, H. (2014, June). A survey on interactive lenses in visualization [DOI: 10.2312/eurovisstar.20141172]. In R. Borgo, R. Maciejewski, & I. Viola (Eds.), *Eurovis – stars: Eurographics conference on visualization, state of the art reports* (pp. 43–62). The Eurographics Association.
- Tsinarakis, G. (2018). Modeling task dependencies in project management using petri nets with arc extensions [DOI: 10.1109/MED.2018.8442472]. *Proceedings of the 26th Mediterranean Conference on Control and Automation (MED'18), June 25–28, 2018, Palma de Mallorca, Spain*, 84–89.
- Undozerov, V. (2022). Dynamic visualization of construction schedules on the example of the spring chart technique [DOI: 10.1007/978-3-030-96206-7_47]. *Proceedings of the 10th International Conference on Computing in Civil and Building Engineering (ICCCBE 2022), August 2022, Cape Town, South Africa*, 449–456.
- Van Peteghem, V., & Vanhoucke, M. (2014). An experimental investigation of metaheuristics for the multi-mode resource-constrained project scheduling problem on new dataset instances [DOI: 10.1016/j.ejor.2013.10.012]. *European Journal of Operational Research*, 235(1), 62–72.
- Vergara Fernandez, E., Rousis, V., et al. (2023). Scheduling tools development to manage CERN accelerators programmed stops and facilities installations [DOI: 10.18429/JACoW-IPAC2023-TUPM073]. *Proceedings of the 14th International Particle Accelerator Conference (IPAC'23)*, 2364–2366.
- Vergara Fernandez, E., Ansel, A., Barberan Marin, M., Bernardini, M., Chemli, S., Coupard, J., Foraz, K., Hay, D., Jimenez, J., Mcfarlane, D., Pedrosa, F., Pirozzi, M., & Tock, J.-P. (2022). Processes and tools to manage CERN programmed stops applied to the second long shutdown of the accelerator complex [DOI: 10.18429/JACoW-IPAC2022-WEPOTK009]. *Proceedings of the 13th International Particle Accelerator Conference (IPAC'22)*, 2048–2051.
- Wang Baldonado, M. Q., Woodruff, A., & Kuchinsky, A. (2000). Guidelines for using multiple views in information visualization [DOI: 10.1145/345513.345271]. *Proceedings of the*

- Working Conference on Advanced Visual Interfaces (AVI 2000)*, May 2000, Palermo, Italy, 110–119.
- Wen, X., Zhang, X., Xing, H., Ye, G., Li, H., Zhang, Y., & Wang, H. (2024). An improved genetic algorithm based on reinforcement learning for aircraft assembly scheduling problem [DOI: 10.1016/j.cie.2024.110263]. *Computers and Industrial Engineering*, 193, 110263.
- Zhang, Y., Li, C., Su, X., Cui, R., & Wan, B. (2023). A baseline-reactive scheduling method for carrier-based aircraft maintenance tasks [DOI: 10.1007/s40747-022-00784-9]. *Complex and Intelligent Systems*, 9(1), 367–397.

Appendix A

Evaluation Instrument

This appendix contains the full set of materials used to administer and record the evaluation protocol described in Chapter 6. Each section corresponds to one phase of the session or to one operational artifact that supports it.

A.1 Introduction

Hello, good morning/afternoon. Thank you very much for reserving some of your time to help us today.

For this session, the prototype runs locally on the moderator’s laptop, which is the device you will be using throughout the session.

A.2 Informed Consent Form

The informed consent form is presented as follows.

A.2.1 Introduction

You are invited to participate in a research study to evaluate a prototype planning tool for CERN interventions. The tool integrates a reusable template engine, a constraint-programming optimizer, and an interactive review interface. This experiment is part of a Master’s Dissertation aimed at assessing the tool’s usability and effectiveness. Your participation will help us understand how effectively users can compose, optimize, and refine intervention schedules with the tool.

A.2.2 Voluntary Participation

Participation in this experiment is entirely voluntary. You are free to decide whether or not to take part. If you choose to participate, you may withdraw at any time during the experiment without any consequences or need to provide a reason. To withdraw, simply inform the researcher, and any data collected from you up to that point will be deleted and not used in the study.

A.2.3 Experiment Overview

As a participant, you will be asked to perform a set of short tasks with the prototype and to think aloud while doing so. The tasks cover the main features of the tool: creating templates, assembling a schedule from templates, running the optimizer, reviewing the result, and exporting to Microsoft Project. After the tasks, you will rate each task on two short Likert scales and answer a short questionnaire about your experience. If you are a planning coordinator, you will also be asked a few questions comparing your current Microsoft Project workflow with the prototype.

A.2.4 Data Collection

We will register your responses to the pre-test, task ratings, and post-test questionnaires, your answers to the conditional workflow comparison questionnaire, and moderator notes taken during the session. With your explicit consent, we may also record the screen and/or voice during the session. Demographic information is collected only for the purpose of contextualizing the results.

A.2.5 Data Usage and Confidentiality

All data collected during this experiment will be used solely for research purposes as part of this Dissertation. Your personal information and responses will remain confidential and will be anonymized. Only the researcher will have access to the raw data, which will be securely stored and deleted after the study is complete.

A.2.6 Contact Information

If you have any question about the experiment or your participation, please contact: Manuel Carlos Ramos Alves, up201906910@up.pt or manuel.carlos.ramos.alves@cern.ch, Master in Informatics and Computing Engineering student at Faculdade de Engenharia da Universidade do Porto.

A.2.7 Consent

I have read and understood the information provided above. I understand that my participation is voluntary, that I can withdraw at any time during the experiment without any consequences or need to provide a reason. I consent to participate in this experiment, including the collection of demographic data and the provision of feedback for system improvements.

Name:

Signature:

Date:

A.3 Pre-test Questionnaire

I would like to start by asking you a few short questions to understand your background. All questions are optional — feel free to skip any you prefer not to answer.

1. **Age.**
2. **Gender.**
3. **Role at CERN.**
4. **Years at CERN.**
5. **Years of Microsoft Project use.**
6. **Familiarity with the CERN planning workflow.** 5-point Likert scale: 1 = not familiar, 5 = very familiar.
7. **Familiarity with the AD/ELENA facility and its intervention planning.** 5-point Likert scale: 1 = not familiar, 5 = very familiar. (Asked because the evaluation tasks use AD/ELENA projects; this rating allows post-hoc analysis of whether prior facility knowledge influences task performance.)
8. **Which tools or techniques do you currently use for planning your work?**

A.4 Task Instructions and Expected Outcomes

I will now ask you to perform eight short tasks in the prototype. For each one I will read the instruction aloud; please tell me whatever comes to your mind as you work, including any doubts or frustrations. For every task, the *expected outcome* on the right is held by the moderator and is used to score task completion — it is not shown to the participant. Tasks are always attempted in the numeric order below.

T1 — Open and inspect a template. *Instruction:* “Navigate to the Templates page and open the template named *AD ring (YETS)*. Inspect both the Gantt view and the table view, switching between the two. Tell me which tasks are summary tasks and which are leaf tasks. Find the duration and the responsible group of the task *Magnets inspection*. Finally, hover over one of the dependency arrows and describe what it represents.”

Expected outcome: the participant correctly identifies summary vs. leaf tasks in both views (e.g., *AD Sect 1B* as a summary, *Magnets inspection* as a leaf); reports the correct duration and group for *Magnets inspection*; correctly interprets the dependency type (FS/SS/FF/SF) from the arrow and tooltip.

T2 — Create a template from an existing project. *Instruction:* “Navigate back to the Templates page and click *Import from Project*. Import the project *AD-ELENA_YETS23-24* from the facility *AD/ELENA*. Within the imported project tree, expand the summary *YETS 23-24* to reach its child summary *ELENA works*. Drag *ELENA works* to include its entire subtree in the selection (this also brings *YETS 23-24* in as the template’s root summary). Deselect *Install new SEM 0225* so that it is not part of the new template. Name the template *ELENA works (YETS)* and confirm.”

Expected outcome: the project *AD-ELENA_YETS23-24* is loaded; the subtree under *ELENA works* (with *YETS 23-24* as its parent) is selected; *Install new SEM 0225* is not part of the final template; the template *ELENA works (YETS)* is created and visible in the Templates list.

T3 — Edit a template. *Instruction:* “Open the template you just created in T2 (*ELENA works (YETS)*). Rename the root summary *YETS 23-24* to *YETS*. Change the duration of *Ion Source maintenance* to ten days. Select the task *STEP cabling* and delete it. Set a Start-No-Earlier-Than (SNET) constraint on *ELENA run with Hminus* with the constraint set to *Day 1 at 08:00* (template-relative). Use undo to revert the last action and then redo it to verify the undo/redo stack. Add a finish-to-start dependency between *Stop of ELENA run with Hminus* and *Ion Source maintenance*. Finally, assign the group *BE-OP* to the task *Extract SEM 0225*.”

Expected outcome: the root summary is renamed from *YETS 23-24* to *YETS*; the duration of *Ion Source maintenance* is set to 10 days; *STEP cabling* is selected and removed from the template; the SNET constraint is saved with Day 1 at 08:00 and the constraint marker on the task changes from the ASAP icon to the SNET icon; undo and redo behave as expected; the new FS dependency is visible as an arrow from *Stop of ELENA run with Hminus* to *Ion Source maintenance*; the group *BE-OP* appears on the *Extract SEM 0225* task in the editor.

T4 — Create a schedule and instantiate templates. *Instruction:* “Navigate to the Schedules page and create a new schedule named *YETS27-28* in facility *AD/ELENA*, starting on *1 November 2027 at 08:00*. Then add a single instance of template *ELENA works (YETS)* (the template you created in T2). Open the Batch Instantiate dialog and add three instances of template *Power Cuts*, with find pattern *EBD7*, replace start number 8, count 3, interval 0 days, and Chain instances set to *None*.”

Expected outcome: the schedule is created in facility *AD/ELENA* with start date 1 November 2027 08:00; one instance of *ELENA works (YETS)* is added and its tasks are visible in the Gantt; three instances of *Power Cuts* are created and named *Power Cuts (EBD8)*, *Power Cuts (EBD9)*, and *Power Cuts (EBD10)*; no FS chain is created between consecutive instances; the find-and-replace substitution is applied to the task names inside each instance (e.g., *AUG tests + EBD8*49, EBD8*14*).

T5 — Add an inter-instance dependency and capacity limits. *Instruction:* “Open the Optimization panel on the right side of the schedule view (settings). Add a finish-to-start dependency from *Power Cuts (EBD10)* to *ELENA works (YETS)* (instance-to-instance). Then configure two

capacity limits: a location concurrency limit of one on *Sector 45*, and a group concurrency limit of one on the team *BE-OP*. Leave the constraint date empty for both limits, so the constraints apply across the whole schedule. Click *Save constraints* to persist the configuration. Finally, looking at the schedule before running the optimizer, identify one location or group capacity violation that is already happening with the current dates and describe it aloud.”

Expected outcome: the Optimization panel is opened on the right; the inter-instance dependency is visible as an arrow from *Power Cuts (EBD10)* to *ELENA works (YETS)*; both capacity limits (*Sector 45* location, *BE-OP* group) are saved (via *Save constraints*) without a constraint date, so they apply across the whole schedule and are visible in the Optimization panel; the participant identifies at least one pre-optimization violation on either the *Sector 45* or the *BE-OP* capacity row (e.g., multiple *Power Cuts* overlapping in *Sector 45*, or several *BE-OP* tasks overlapping) and explains aloud which limit is being breached.

T6 — Lock a task, run the optimizer, and interpret the result. *Instruction:* “Lock the task *Long pumping LNE02* (inside the *ELENA works (YETS)* instance) so that the optimizer cannot change its dates. Then run the optimizer. When the result appears, scroll down inside the Optimization panel to find the Schedule Changes list. Inspect the Schedule Changes list, the KPI cards, and the ghost overlay. Check whether the location and group concurrency limits you set in T5 are respected in the optimized schedule. Tell me aloud what changed, which tasks moved, and what caused the main movement.”

Expected outcome: the lock icon appears on *Long pumping LNE02* and the optimizer leaves its dates unchanged in the result; the optimizer completes and produces a hierarchical Schedule Changes list grouped by instance (e.g., one entry per *Power Cuts* instance and one for *ELENA works (YETS)*); the participant identifies the total count of changes and at least one moved task by name; the participant offers a reasoning linking the main movement to a dependency (intra-template or the inter-instance FS *Power Cuts (EBD10)* → *ELENA works (YETS)*) or to one of the capacity limits (*Sector 45* location or *BE-OP* group); the participant confirms that the concurrency limits are respected (no two tasks in *Sector 45* overlap; no two *BE-OP* tasks overlap).

T7 — Edit & Review the result. *Instruction:* “Inside the Optimization panel, enter Edit & Review mode. On the Gantt, manually drag *Replace BTV cameras* (within *ELENA works (YETS)*) so that its start is positioned *before* the finish of *Stop of ELENA run with Hminus*. In the changes panel inside the Optimization panel, verify that a potential violation has been flagged: the finish-to-start dependency from *Stop of ELENA run with Hminus* to *Replace BTV cameras* is now broken. To resolve the violation, expand *ELENA works (YETS)* in the changes list, locate the *Replace BTV cameras* entry (which now carries a *Manual* badge), and click the *X* on the *Manual* badge to revert the manual drag.”

Expected outcome: Edit & Review mode is entered from inside the Optimization panel; the manual drag of *Replace BTV cameras* is performed and places its start before the finish of *Stop of ELENA run with Hminus*; in the changes panel inside the Optimization panel, a violation alert

appears with a message indicating that the FS dependency from *Stop of ELENA run with Hminus* is violated (lag 1); the participant locates *Replace BTV cameras* under *ELENA works (YETS)* in the expanded changes list, identifies the *Manual* badge, and clicks its *X* to revert the drag; after the revert, the violation alert disappears.

T8 — Accept and export. *Instruction:* “Save the accepted result of your Edit & Review session. Export the schedule to Microsoft Project. Open the exported file on this laptop and confirm that the tasks, their dependencies, and the constraints are preserved in Microsoft Project.”

Expected outcome: the Edit & Review result is saved; the export produces a file (.xml) that opens in Microsoft Project without errors; the tasks, dependencies, and constraints are recognizable in the exported file.

A.5 Task Ratings Grid

After completing all eight in-test tasks, the participant fills a single consolidated grid rating each task on two 5-point Likert scales. A single optional free-text field at the bottom of the grid collects any comment the participant wishes to add about specific tasks.

For each of the eight tasks, please indicate how difficult you found it and how useful the feature it exercised was.

Task	Difficulty					Usefulness				
	1	2	3	4	5	1	2	3	4	5
T1 — Open and inspect a template	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
T2 — Create a template from a project	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
T3 — Edit a template	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
T4 — Create a schedule and instantiate	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
T5 — Inter-instance dep + capacity limits	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
T6 — Lock, optimize, interpret	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
T7 — Edit & Review the result	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
T8 — Accept and export	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

Difficulty: 1 = very easy, 5 = very difficult. *Usefulness:* 1 = not useful, 5 = very useful.

Additional comments (optional). Any specific task you would like to comment on?

A.6 Post-test Questionnaire

The post-test questionnaire is filled in after the task ratings grid.

A.6.1 System Usability Scale (SUS)

The ten items below are administered in the order shown. Each item is rated on a 5-point Likert scale from 1 (strongly disagree) to 5 (strongly agree). The overall SUS score in the [0, 100] range is computed following the procedure described in (Brooke, 1996).

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

A.6.2 Open-ended questions

These questions are asked verbatim. The participant may answer in writing or verbally, in which case the moderator transcribes the answer.

1. How useful would this tool be in your current planning workflow?
2. What were the weakest or most frustrating parts of the experience?
3. What suggestions would you make to improve the tool?
4. What aspect of the tool, if any, surprised you the most during the session — positively or negatively?
5. Is there any feature you expected but could not find?

A.7 Track B Workflow Comparison Questionnaire

Administered orally to planning coordinators immediately after the Track A post-test questionnaire. The moderator reads each question aloud and records the response on their own laptop; the participant does not fill the form in writing.

1. **Confidence in output quality.** How confident are you that the prototype could produce a schedule at least as good as the one you would produce with your current Microsoft Project workflow? (5-point Likert: 1 = not confident, 5 = very confident)
2. **Perceived speed.** How much faster or slower do you expect the prototype to be when building a schedule, compared to your current Microsoft Project workflow? (5-point Likert: 1 = much slower, 3 = about the same, 5 = much faster)
3. **Behavior under complexity.** As schedules become more complex (more template instances, more dependencies, tighter constraints), how do you expect the benefit of the prototype to change? (5-point Likert: 1 = the advantage disappears, 3 = unchanged, 5 = the advantage grows considerably)
4. **Where the prototype helps most.** Which aspects of your current Microsoft Project workflow do you expect the prototype to improve the most?
5. **Where the prototype does not help.** Which aspects of your current workflow do you not expect the prototype to improve, or in which the prototype may even perform worse?
6. **Adoption.** Would you consider adopting the prototype for a real intervention plan? Why or why not?
7. **Conditions for replacement.** What would you need from the prototype to feel comfortable replacing your current Microsoft Project workflow with it?

Appendix B

Collected Data

This appendix reproduces the per-participant data collected in Tracks A and B, together with the aggregated results of the Track C benchmark, anonymized as described in Chapter 7. Track A participants are identified by the codes P1 to P20 and the Track B respondents by B1 to B7; roles are reported at the level of function in order to preserve anonymity.

B.1 Participant Demographics

Table B.1 lists the background of the twenty Track A participants, including which of them are planning coordinators. The bimodal Microsoft Project experience and the uniformly low AD/E-LENA familiarity discussed in Section 7.1 are visible in the right-hand columns.

B.2 Planning Tools Currently Used

Table B.2 reports the tools each participant uses in their current planning practice, from the open pre-test question. Microsoft Project dominates, often alongside spreadsheets; a dash denotes no tool reported.

B.3 Task Performance

Table B.3 gives the time spent on each task by each participant. Every participant completed every task, so the completion column is uniformly positive and is omitted; the per-task error and assist counts are reported in Section 7.2.1.

Table B.4 gives the subjective difficulty and perceived usefulness each participant assigned to every task.

Table B.1: Demographic and background data for the twenty Track A participants. Group: Coord. denotes a planning coordinator and Non a non-planning member. CERN and MSP are years at CERN and years of Microsoft Project experience; Plan. is self-rated familiarity with the CERN planning workflow (1–5) and AD/EL. is self-rated familiarity with the AD/ELENA facility (1–5).

ID	Group	Role	Gender	Age	CERN	MSP	Plan.	AD/EL.
P1	Coord.	Facility Coordinator	M	58	33	5.5	5	1
P2	Coord.	Project Coordinator	F	27	2.5	4	5	1
P3	Coord.	Technical Coordinator	M	48	24	5	4	4
P4	Coord.	Deputy Technical Coordinator	M	61	24	31	5	1
P5	Coord.	Project Manager	M	25	3	4	2	1
P6	Coord.	Surface Activities Coordinator	M	46	26	12	4	1
P7	Non	Group Leader	M	51	25	5	2	1
P8	Coord.	Deputy Project Leader	M	55	30	35	4	1
P9	Coord.	Facility Coordinator	M	53	28	4	4	1
P10	Non	Radiation Protection Officer	F	32	7	1	3	1
P11	Coord.	Technical Coordinator	M	56	29	29	5	1
P12	Coord.	Facility Coordinator	M	27	0.5	3	4	1
P13	Coord.	Technical Coordinator	M	50	19	20	4	1
P14	Coord.	Planning Engineer	F	28	2.5	3.5	5	1
P15	Coord.	Planning Officer	M	28	2	2	5	1
P16	Coord.	Coordination Team Member	M	62	18	2	3	1
P17	Non	Electrical Engineer	M	29	4	4	5	2
P18	Non	Section Leader	M	49	20	8	4	1
P19	Non	Mechanical Engineer	F	31	2	1	2	1
P20	Non	Project Engineer	F	51	24	10	4	2

Table B.2: Tools and techniques currently used by each participant for planning.

ID	Tools currently used
P1	Microsoft Project (reusing and editing past plannings)
P2	Microsoft Project, Excel
P3	Excel (daily calendar), Power BI (space occupancy)
P4	Microsoft Project
P5	Merlin, Microsoft Project
P6	Merlin, Excel, Monday.com, Microsoft Project
P7	—
P8	Collaboration with schedule-office experts
P9	Microsoft Project, Outlook, Excel
P10	—
P11	Microsoft Project, GDPM, Excel
P12	Microsoft Project
P13	Microsoft Project, Mac Gantt Project
P14	Microsoft Project
P15	Microsoft Project, MS Office, communication tools
P16	Microsoft Project
P17	Microsoft Project, Scheduling Tools
P18	—
P19	—
P20	Microsoft Project, Excel

Table B.3: Time on task in minutes for tasks T1–T8, with the per-participant total.

ID	T1	T2	T3	T4	T5	T6	T7	T8	Total
P1	2.0	2.0	4.0	5.0	5.0	4.0	2.0	2.0	26.0
P2	1.5	2.0	3.5	3.0	3.0	2.0	4.0	2.0	21.0
P3	2.0	3.0	4.0	4.0	3.0	3.0	4.0	2.0	25.0
P4	1.5	4.0	5.0	5.0	3.0	2.5	4.5	1.0	26.5
P5	1.5	3.5	4.0	3.0	2.0	4.0	1.5	2.0	21.5
P6	2.5	3.5	7.0	4.5	3.5	3.0	3.0	2.0	29.0
P7	2.0	1.5	4.5	4.5	3.0	2.0	3.0	1.5	22.0
P8	4.0	2.0	4.0	5.5	3.5	3.5	3.5	1.5	27.5
P9	1.5	2.0	4.0	4.0	5.5	4.0	3.5	1.5	26.0
P10	1.5	2.0	3.0	5.5	3.5	3.0	3.0	2.0	23.5
P11	1.5	1.5	3.5	5.0	3.0	1.5	3.0	1.5	20.5
P12	1.0	1.0	3.0	3.0	2.5	4.0	1.5	1.0	17.0
P13	1.5	2.5	5.0	5.0	5.0	4.5	3.0	1.5	28.0
P14	1.5	2.0	4.0	5.0	3.5	2.0	3.0	1.0	22.0
P15	1.5	1.5	4.0	5.0	4.0	5.0	2.5	2.0	25.5
P16	3.0	2.0	5.0	4.0	4.0	3.0	3.5	1.5	26.0
P17	1.5	1.5	3.0	4.0	3.5	2.0	3.0	1.5	20.0
P18	1.5	1.5	3.5	4.0	3.5	3.0	2.0	1.5	20.5
P19	1.5	1.5	3.5	4.0	3.0	2.5	2.0	1.0	19.0
P20	2.0	2.0	4.0	4.0	3.0	3.0	2.5	1.0	21.5

Table B.4: Per-task ratings for each participant, shown as difficulty / usefulness, each on a five-point scale (difficulty: 1 = very easy, 5 = very hard; usefulness: 1 = not useful, 5 = very useful).

ID	T1	T2	T3	T4	T5	T6	T7	T8
P1	1/5	1/5	1/5	2/5	2/4	1/4	1/4	1/5
P2	1/5	1/4	2/5	3/5	1/4	1/5	2/5	1/5
P3	2/5	2/4	2/5	4/4	3/3	1/5	4/4	1/5
P4	1/4	1/4	1/4	3/5	2/5	1/5	2/3	1/5
P5	1/5	2/5	2/5	1/5	1/4	1/5	1/5	1/5
P6	1/5	1/4	2/4	2/3	2/4	2/5	2/5	1/5
P7	1/4	2/4	1/4	3/4	3/4	3/4	3/4	2/4
P8	2/5	2/5	1/5	2/4	1/4	3/3	3/4	1/5
P9	1/5	2/5	2/4	2/5	2/5	1/5	3/5	1/5
P10	1/5	1/5	1/5	2/5	2/5	1/5	1/5	1/5
P11	1/5	1/5	1/5	1/5	2/5	1/5	2/5	1/5
P12	1/4	2/5	2/4	1/5	2/4	1/5	2/4	1/4
P13	1/5	1/3	2/5	1/5	1/5	2/4	2/5	1/5
P14	1/5	1/5	1/5	2/5	1/5	1/5	2/5	1/5
P15	2/4	1/4	2/4	3/4	2/5	1/5	1/5	1/4
P16	1/5	1/5	1/5	1/4	2/4	2/3	2/3	1/5
P17	1/5	1/5	1/5	1/4	3/3	1/5	2/3	1/5
P18	1/5	1/5	1/5	2/4	2/4	1/5	2/4	1/5
P19	1/5	1/5	1/5	1/5	1/5	2/5	1/5	1/5
P20	1/5	1/5	1/5	2/5	1/5	2/5	1/5	1/5

B.4 System Usability Scale Responses

Table B.5 gives each participant's response to the ten SUS items. Odd-numbered items are positively worded and even-numbered items negatively worded; the resulting per-participant scores are shown in Fig. 7.7.

Table B.5: Per-item System Usability Scale responses (1–5) for each participant.

ID	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	SUS
P1	5	1	5	1	5	1	5	1	5	1	100
P2	5	1	4	2	5	1	4	1	5	1	92.5
P3	4	2	3	4	5	1	4	1	2	1	72.5
P4	3	1	4	1	5	1	4	2	5	1	87.5
P5	4	1	5	2	5	1	4	1	5	2	90
P6	4	4	3	5	5	2	4	1	3	4	57.5
P7	4	3	4	3	4	2	4	2	4	4	65
P8	3	1	4	2	4	2	5	1	3	2	77.5
P9	3	1	5	2	3	1	3	1	5	1	82.5
P10	4	1	5	2	4	1	5	2	3	2	82.5
P11	4	1	5	1	4	1	5	1	3	1	90
P12	2	2	4	3	5	1	4	2	4	2	72.5
P13	2	1	5	4	5	1	4	1	3	1	77.5
P14	4	1	5	4	5	1	5	1	4	1	87.5
P15	4	1	4	1	5	2	4	2	3	2	80
P16	3	1	5	3	4	1	5	3	5	5	72.5
P17	4	2	4	3	4	1	4	3	4	1	75
P18	3	1	4	2	5	1	4	1	4	1	85
P19	4	1	5	1	5	1	5	1	4	1	95
P20	4	1	4	2	5	1	5	1	3	1	87.5

B.5 Open-ended Feedback Themes (Track A)

The free-text responses and think-aloud commentary of Track A were analyzed thematically. Table B.6 groups the recurring themes, the participants who raised each, and a short description of what the theme refers to, separating the strengths the participants highlighted from their suggestions for improvement. This is the structured basis for the synthesis in Section 7.2.5.

B.6 Track B: Workflow Comparison Responses

Seven of the fourteen coordinators completed the Track B questionnaire; they are identified as B1 to B7. Table B.7 gives their responses to the three scaled questions, and Table B.8 summarizes the four open-ended questions thematically.

B.7 Algorithmic Benchmark Data

This section reproduces the aggregated results of the Track C benchmark (Section 7.4), one table per experiment (Tables B.9, B.10, B.11, and B.12). Each row is a configuration; the time reported

Table B.6: Thematic summary of the Track A open-ended feedback, with the participants who raised each theme.

Theme	Participants	Description
<i>Strengths highlighted</i>		
Templates and optimization	P1, P3, P5, P9, P11, P15, P19	The two core features were singled out as the most valuable, both for building the initial plan and for revising it efficiently.
Ease of first use	P4, P14, P17	The tool was found approachable on first contact, without training, which several noted is unusual for scheduling software.
Import and integration	P9, P13	Importing an existing project and the round trip with Microsoft Project were appreciated.
Constraint visualization	P2	The graphical notation for dependencies and constraints was found clear and easy to read.
<i>Suggestions for improvement</i>		
Optimizer transparency	P1, P7, P9, P15	A clearer view of what the optimizer changed, including a before-and-after comparison and a progress indicator while it runs.
Schedule-level undo/redo	P7, P13, P14	Undo and redo within the schedule view, matching what is already available in the template editor.
Accept and revert clarity	P7, P9	Clearer signposting of how to revert a change and of the accept-or-discard workflow.
Gantt overview and zoom	P7, P11, P15, P20	A broader view of the schedule, with zoom, a critical-path view, and customizable columns.
Search and navigation	P5, P10, P17, P20	Faster search and easier navigation back to the template library.
Export formats	P2, P3	Export to native .mpp, Excel, and PDF in addition to Microsoft Project XML.
Template terminology	P2	Clearer naming and explanation of batch instantiation and template concepts.
Task hierarchy editing	P3	Reordering tasks within the hierarchy and moving them to a different parent.
Richer optimization controls	P11, P14	More constraint types, task priorities, and resource-based optimization options.
Onboarding and help	P5, P8, P14, P16, P17	Better discoverability of features on first use, and an in-app help or FAQ section.
Dependency and date clarity	P3, P16, P20	Clearer dependency labels (such as “FS”) and an easier way to enter the relative dates of template tasks.
Confirmation and safeguards	P6, P11	A confirmation prompt before irreversible actions, and an optional human-check step before the optimization is applied.
Settings and layout consistency	P4, P8	A consistent path for applying changes, and a tidier settings panel and screen layout.
Lock and filter visibility	P12	Seeing at a glance which tasks are locked, and filtering the task list.
Collaboration and audit trail	P6	Multi-user editing and a record of who changed what, when, and why.
Localization	P16	A French-language version of the interface.

Table B.7: Track B responses to the three scaled questions (1–5). Confidence: that the prototype could produce a schedule at least as good as the manual one; Speed: expected speed relative to the current workflow; Benefit with complexity: expected change as schedules grow more complex.

ID	Confidence	Speed	Benefit with complexity
B1	5	3	3
B2	3	4	2
B3	5	4	5
B4	5	3	3
B5	4	4	4
B6	3	4	3
B7	4	4	3

Table B.8: Thematic summary of the Track B open-ended responses (seven coordinators).

Question	Summary of responses
Aspects improved most	Templates and the initial setup of a plan: reusing a known structure without copy-paste errors, and a global view of constraint compliance for large stops. Raised by six of the seven; the seventh cited resource management.
Aspects not improved	The optimization of non-standard work: implicit and partial constraints that coordinators hold as tacit knowledge are difficult to formalize. Raised by five of the seven; one cited the creation of new tasks from scratch, and one left this blank.
Willingness to adopt	All seven would consider adopting the tool: four without qualification, and three scoped to small projects or to specific checks as a first step.
Conditions to replace Microsoft Project	A complete, validated template library for each facility (four coordinators) and parity with the breadth of mature Microsoft Project features (two); one considered the prototype already sufficient.

is the mean and the makespan the median of the three runs, given separately for the two templates. The status conventions are defined in each table's note.

Table B.9: Scalability versus problem size, by inter-instance dependency structure.

Inter-deps	Instances	Tasks	Template 1		Template 2	
			m.s.t.	m.m.	m.s.t.	m.m.
none	2	194	40	127	27	112
none	4	388	69	127	73	112
none	6	582	95	127	115	112
none	8	776	127	127	176	112
none	10	970	142	127	152	127 [†]
none	12	1164	178	127	198	127 [†]
none	16	1552	—	—	—	—
none	20	1940	—	—	—	—
chain	2	194	40	255	24	227
chain	4	388	50	511	35	455
chain	6	582	58	767	56	682
chain	8	776	71	1022	71	911
chain	10	970	78	1278	205	1160 [‡]
chain	12	1164	76	1535	94	1417
chain	16	1552	139	2191 [‡]	154	1933
chain	20	1940	166	2707 [†]	172	2450

m.s.t. = mean solve time (s); m.m. = median makespan (d). [†] feasible (optimality not proven); [‡] the three runs mixed optimal and feasible; — no solution within the limit.

Table B.10: Time granularity. The optimal makespan is identical across granularities.

Slot	Instances	Tasks	Template 1		Template 2	
			m.s.t.	m.m.	m.s.t.	m.m.
15 min	2	194	33	127	41	112
15 min	4	388	74	127	128	112
15 min	6	582	115	127	148	112 [‡]
15 min	8	776	151	127 [‡]	199	127 [‡]
15 min	10	970	132	127 [‡]	—	—
30 min	2	194	32	127	43	112
30 min	4	388	63	127	119	112 [‡]
30 min	6	582	92	127	164	112 [‡]
30 min	8	776	148	127	290	112 [‡]
30 min	10	970	91	127	—	—
60 min	2	194	33	127	27	112
60 min	4	388	61	127	73	112
60 min	6	582	85	127	117	112
60 min	8	776	106	127	181	112
60 min	10	970	143	127	145	127 [†]

m.s.t. = mean solve time (s); m.m. = median makespan (d). †/‡/— as in Table B.9.

Table B.11: Warm-start, by Phase 1 mode.

Phase 1	Instances	Tasks	Template 1		Template 2	
			m.w.t.	m.m.	m.w.t.	m.m.
off	2	194	69	127	47	112
off	4	388	105	127	123	112
off	6	582	124	127	145	112 [‡]
off	8	776	148	127	218	112 [‡]
off	10	970	217	127	223	127 [†]
day	2	194	65	127	74	112
day	4	388	109	127	128	112 [‡]
day	6	582	158	127	185	112 [‡]
day	8	776	261	127	302	112 [‡]
day	10	970	341	127	394	127 [†]
week	2	194	33	132	39	112
week	4	388	162	127	81	112
week	6	582	175	190 [†]	179	112 [‡]
week	8	776	265	190 [‡]	250	112 [‡]
week	10	970	404	190 [†]	313	282 [‡]

m.w.t. = mean wall-clock time (s); m.m. = median makespan (d). †/‡/— as in Table B.9.

Table B.12: Concurrency limits, for $N = 2$ independent template instances (194 leaf tasks, no inter-instance chain), by constraint set and limit.

Constraint	Limit	Template 1		Template 2	
		m.s.t.	m.m.	m.s.t.	m.m.
none (baseline)	—	31	127	30	112
location	1	136	326 [†]	147	326 [†]
location	2	138	162 [†]	259	165 [†]
location	3	180	127 [‡]	105	113 [†]
location	5	60	127	100	112 [†]
team	1	126	208 [†]	100	197 [†]
team	2	115	132 [†]	89	112 [†]
team	3	85	127	62	112
team	5	46	127	32	112

m.s.t. = mean solve time (s); m.m. = median makespan (d). [†]/[‡]/— as in Table B.9.

Table B.13: Makespan under growing inter-instance coupling, for $N = 4$ independent sparse-template instances (388 leaf tasks). Each of the K links is a finish-to-start dependency between randomly chosen tasks of two different instances; each count is solved three times, with a fresh random choice of links.

Inter-instance deps K	m.s.t.	m.m.
0	77	112
4	29	257
8	34	300
16	23	386
32	24	386
64	25	386

m.s.t. = mean solve time (s); m.m. = makespan (d), mean over three runs; all runs optimal. At $K = 8$ the three runs gave 257, 257, and 386 days (mean 300); at every other count they were identical.

Table B.14: Makespan under team and location concurrency limits applied separately and together, for $N = 2$ independent sparse-template instances (194 leaf tasks). Each configuration is solved three times.

Constraint	Limit	m.s.t.	m.m.
none (baseline)	—	28	112
team	1	107	197 [†]
team	2	94	112 [†]
team	3	66	112
location	1	158	326 [†]
location	2	274	165 [†]
location	3	104	113 [†]
both	1	239	327 [†]
both	2	278	170 [†]
both	3	102	113 [†]

m.s.t. = mean solve time (s); m.m. = makespan (d), identical across the three runs. [†] feasible (optimality not proven).