FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

DOCTORAL THESIS

Mgr. Marek Vlk

Optional Activities in Scheduling

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the doctoral thesis:	Prof. RNDr. Roman Barták, Ph.D.
Study programme:	Theoretical Computer Science and Artificial Intelligence

Prague 2021

I would like to thank my supervisor Prof. RNDr. Roman Barták, Ph.D. for his valuable guidance and support during my research and studies. I would also like to thank those who directly or indirectly cooperated with me and thus contributed to the results, namely, Zdeněk Hanzálek, Přemysl Šůcha, Jiří Švancara, István Módos, Ondřej Benedikt, Antonín Novák, Arnaud Malapert, Philippe Laborie, and Emmanuel Hebrard. Special thanks go to my family and friends who were supportive all along.

Title: Volitelné aktivity v rozvrhování

Author: Mgr. Marek Vlk

Department: Katedra teoretické informatiky a matematické logiky

Supervisor: Prof. RNDr. Roman Barták, Ph.D., Katedra teoretické informatiky a matematické logiky

Abstract: Rozvrhování přiděluje omezené zdroje na aktivity tak, aby byly splněny určité podmínky a optimalizovány konkrétní cíle. Aktivity, které mají být provedeny, jsou obvykle známé nebo jsou určeny předem ve fázi plánování. Pro zlepšení pružnosti rozvrhovacích systémů byl vyvinut koncept volitelných aktivit. Volitelné aktivity jsou ty aktivity, o jejichž výskytu ve výsledném rozvrhu se má rozhodnout. Spíše než určovat, které aktivity je třeba vykonat, a rozvrhovat je, ve dvou po sobě jdoucích fázích, lze pružnost a účinnost významně zlepšit, pokud je výběr aktivit i přidělení času integrován do stejného řešiče. Takový přístup byl implementován u několika řešičů programování s omezujícími podmínkami a projevil se skvělým výkonem na řadě rozvrhovacích problémů. V této práci aplikujeme koncept volitelných aktivit na problémy rozvrhování, které na první pohled nezahrnují volitelné aktivity, jako je problém rozvrhování výroby s nepřekrývajícími se seřízeními závislými na sekvenci vykonávaných úloh, ale také na problémy mimo doménu rozvrhování, jako je problém hledání cest více agentů a jeho rozšíření zahrnující hrany s různými vahami a kapacitami.

Keywords: Programování s omezujícími podmínkami, hledání cest více agentů, rozvrhování nepřekrývajících se seřízení, společné směrování a rozvrhování

Title: Optional Activities in Scheduling

Author: Mgr. Marek Vlk

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Prof. RNDr. Roman Barták, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: Scheduling allocates scarce resources to activities such that certain constraints are satisfied and specific objectives are optimized. The activities to be executed are commonly known or determined a priori in the planning stage. To improve the flexibility of scheduling systems, the concept of optional activities was invented. Optional activities are those activities whose presence in the resulting schedule is to be decided. Rather than determining which activities need to be executed and scheduling them in two consecutive phases, flexibility and efficiency can be improved significantly when both activity selection and time allocation are integrated within the same solver. Such an approach was implemented in a few Constraint Programming solvers and manifested great performance on multiple scheduling problems. In this thesis, we apply the concept of optional activities to scheduling problems that do not seem to involve optional activities, such as the production scheduling problem with sequence-dependent non-overlapping setups, but also on problems beyond the scheduling domain, such as the multi-agent path finding problem and its extension with weighted and capacitated arcs.

Keywords: Constraint Programming, Multi-agent Path Finding, Non-overlapping Setups Scheduling, Joint Routing and Scheduling

Contents

Introduction **3**

1 Background and Terminology **5**
- 1.1 Constraint Programming . . . 5
- 1.2 Integer Linear Programming . . . 5
- 1.3 Boolean Satisfiability and Satisfiability Modulo Theories . . . 6
- 1.4 Scheduling in IBM CP Optimizer . . . 7

2 Optional Activities for Non-overlapping Setups **8**
- 2.1 Problem Specification . . . 9
  - 2.1.1 Related Work . . . 9
  - 2.1.2 Formal Definition . . . 10
  - 2.1.3 Complexity . . . 10
- 2.2 Integer Linear Programming Model . . . 11
  - 2.2.1 Formulation for a Single Machine . . . 13
  - 2.2.2 Additional Improvements . . . 13
- 2.3 Constraint Programming Models . . . 14
  - 2.3.1 CP1: with Implications . . . 14
  - 2.3.2 CP2: with Element Constraints . . . 15
  - 2.3.3 CP3: with Optional Interval Variables . . . 15
  - 2.3.4 CP4: with Cumulative Function . . . 16
  - 2.3.5 CP5: without Setup Variables . . . 17
  - 2.3.6 Additional Improvements . . . 18
- 2.4 Heuristic Approach . . . 19
  - 2.4.1 Decomposition Phase . . . 19
  - 2.4.2 Improving Phase . . . 19
- 2.5 Experiments . . . 22
  - 2.5.1 Problem Instances . . . 22
  - 2.5.2 Scalability Results . . . 22
  - 2.5.3 Effect of Exact/Heuristic Subproblem Solution . . . 25
  - 2.5.4 Discussion . . . 29
- 2.6 Summary . . . 29

3 Optional Activities in Multi-agent Path Finding **30**
- 3.1 Problem Specification . . . 30
- 3.2 Scheduling Model . . . 31
  - 3.2.1 Single-layer Models . . . 31
  - 3.2.2 Multi-layer Model . . . 37
  - 3.2.3 Algorithm . . . 39
- 3.3 Generalized MAPF . . . 41
  - 3.3.1 Scheduling-Based Approach . . . 42
  - 3.3.2 Heuristic Approach . . . 43
  - 3.3.3 SAT-Based Approach . . . 43
- 3.4 Experiments . . . 43
  - 3.4.1 Implementation Details . . . 44

3.4.2 Problem Instances 44
3.4.3 Results 44
3.5 Summary 48

4 Optional Activities in Time-Sensitive Networking 50
4.1 Related Work 51
4.2 Problem Specification 52
4.2.1 Network Model 53
4.2.2 Application Model 53
4.2.3 Example 55
4.2.4 Scheduling Constraints 55
4.3 Solution 60
4.3.1 CP1: Model with Disjunctions 62
4.3.2 CP2: Model with Interval Variables 62
4.3.3 Basic Implementation Improvements 63
4.3.4 Restricting Queues to one 64
4.3.5 Logic-Based Benders Decomposition 64
4.3.6 Routing Problem Criteria 66
4.4 Baseline Approaches 68
4.5 Experiments 69
4.5.1 Problem Instances 70
4.5.2 Comparison of Schedulability 71
4.5.3 Usage of Queues and Optimality 74
4.5.4 Impact of Various Routing Criteria, Period Sets, and Problem Modifications 76
4.6 Summary 77

Conclusion 79

Bibliography 80

List of Abbreviations 87

List of Publications 88

Introduction

Scheduling aims to allocate scarce resources to activities in order to satisfy certain constraints and optimize specific objectives. The set of activities to be performed is usually given or determined in advance in the planning stage. To increase the flexibility of scheduling systems, the notion of *optional activities* was introduced. An optional activity is such an activity that eventually may or may not be present in the resulting schedule. Rather than planning which activities need to be performed and scheduling them separately, integrating both activity selection and time allocation in the same engine can improve flexibility and efficiency. Indeed, such an approach has been implemented in a few Constraint Programming (CP) solvers and exhibited great success in various scheduling problems. Moreover, as we do in this thesis, the concept of optional activities can be applied to scheduling problems that do not seem to involve optional activities.

We address a scheduling problem that emerges in the production of water tubes of various sizes that require reconfiguration of the machines. The reconfiguration of the machines leads to the notion of sequence-dependent setup times between tasks. These setups are often performed by a single person who cannot serve more than one machine simultaneously, i.e., the setups must not overlap. Although all the tasks in the problem are known and assigned to the machines, the order of the tasks and, therefore, the setups to be performed are not known. It must be decided in what order to execute the tasks on machines and in what order to perform the resulting setups to minimize makespan.

To solve this problem with sequence-dependent non-overlapping setups, we propose an efficient CP model using the optional activities to represent the setups between tasks and imposing the no-overlap constraint over these optional activities. For each pair of potentially consecutive tasks on a machine, an optional activity is created whose presence is entailed from the order of tasks on the machine. The results of this approach and other variants were elaborated in [1] and [2].

We go even further and leverage the concept of optional activities even on problems beyond the scheduling domain. In particular, another problem that we address in this thesis is multi-agent path finding (MAPF), which aims to find a collision-free path for a set of agents. The agents are located at nodes of a graph, they can move over the arcs, and each agent has its destination node. Two agents cannot be at the same node at the same time. The usual setting is that each arc has a length of one, so at any time step, each agent either stays in the node where it is or moves to one of its neighboring nodes.

We suggest modeling the MAPF problem using scheduling techniques. Namely, nodes and arcs are seen as resources. The concept of optional activities is used to model which nodes and arcs an agent will visit. The significant contribution of the scheduling model of MAPF is its capability to include other constraints naturally. We study particularly the problems where the capacity of arcs can be greater than one (more agents can use the same arc at the same time), and the lengths of arcs can be greater than one (moving between different pairs of nodes takes different times). These extensions make the model closer to reality than the original MAPF formulation. This approach and its variants were published in [3] and [4], and the dynamic version of MAPF, where agents appear online, was investigated in [5].

The same idea from our approach to MAPF can also be applied to routing and scheduling in communication networks, where the data flows are periodically transmitted from end-stations to other end-stations via network switches. For real-time critical data traffic, the deterministic behavior and timeliness guarantees are achieved by time-triggered schedules. Calculating such a schedule requires satisfying a lot of constraints, which in practice may be difficult due to the increasing number of data flows in the networks. The routes of flows are usually given or determined separately from the scheduling process in the current approaches. If the routes of flows can be computed jointly with scheduling, more flows can be scheduled.

To solve the problem of joint routing and scheduling, we propose a model with optional activities that represent the transmissions of data frames on communication links through which the data will be forwarded. Besides, one of the specifics of time-sensitive networks is the usage of queues. However, the frames of critical flows must not be stored in the same queue simultaneously. We model this by optional activities representing waiting of frames in queues and imposing the no-overlap constraint on these optional activities. Hence, the presence of optional activities is selected according to where the flows will be forwarded, and these activities, in turn, select one of its sub-activities to be present according to in which queue the frame will be stored. This novel CP model brought a significant increase in the number of successfully scheduled instances (referred to as schedulability) over the currently used solving methods. These results have been published in [6]. We also developed an efficient heuristic algorithm that solves the scheduling problems on large-scale networks (but disregards alternative routing options), which is published in [7]. Besides, we investigated a possibility to improve schedulability and throughput by enhancing the hardware of switches [8].

The rest of the thesis is organized as follows. In the first chapter, we briefly introduce some important terminology. In Chapter 2, we solve the production scheduling problem with sequence-dependent non-overlapping setups. In Chapter 3, we tackle the problem of multi-agent path finding and its extended variant with weighted and capacitated arcs. Finally, in Chapter 4, we address the problem of joint routing and scheduling of time-triggered traffic in time-sensitive networks.

1. Background and Terminology

In this chapter, we first describe specific exact techniques for solving combinatorial optimization problems, and then we focus on the modeling formalism and the solver that we use in this thesis.

1.1 Constraint Programming

A *constraint satisfaction problem* (CSP) [9] is a triple (X, D, C), where

- $X = \{x_1, ..., x_n\}$ is a set of variables,
- $D = \{D_1, ..., D_n\}$ is a set of nonempty domains of values for each variable,
- $C = \{C_1, ..., C_m\}$ is a set of constraints.

Each variable x_i can take on the values from the domain D_i. Constraint C_j involves some subset of variables and specifies the allowable combinations of values for that subset of variables.

A state of the problem is an *assignment* (or an instantiation) of values to some or all of the variables $(x_i = a_i, x_j = a_j, ...)$. An assignment that does not violate any of the constraints is called *feasible* or *consistent*. An assignment that includes all variables is called *complete*. A *solution* of a CSP is a complete and feasible assignment.

A *constraint optimization problem* (COP) is an extension of a CSP that includes an objective function. An *optimal* solution to a minimization (maximization) COP is a feasible solution that minimizes (maximizes) the value of the objective function.

Constraint Programming (CP) is a powerful paradigm for solving CSPs and COPs that makes use of a wide range of techniques from artificial intelligence, computer science, databases, programming languages, and operations research [10]. CP solvers typically use a form of search. The most used techniques are variants of *backtracking* to assign values to variables and *constraint propagation* to prune the domains of variables.

In general, CP solvers usually find a high-quality solution quickly but sometimes struggle to prove optimality or infeasibility. The CP solver used in this thesis, which is the state-of-the-art proprietary CP solver, primarily for scheduling applications, is IBM CP Optimizer [11]. Its success stems mainly from particular global constraints modeling unary (disjunctive) resources and efficient filtering algorithms for these constraints [12].

1.2 Integer Linear Programming

An *integer linear programming* (ILP) problem is a mathematical optimization program in which the variables are restricted to be integers, the objective function and the constraints are linear. More precisely, an ILP can be expressed as:

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \le \mathbf{b} \\ & \mathbf{x} \ge \mathbf{0} \\ & \mathbf{x} \in \mathbb{Z}^n \end{aligned}$$

where $\mathbf{c}$ and $\mathbf{b}$ are vectors and $\mathbf{A}$ is a matrix. An ILP can be conceived as a special case of a COP, but every COP can be converted to an equivalent ILP and vice versa. Note that relaxing the constraint $\mathbf{x} \in \mathbb{Z}^n$ to $\mathbf{x} \in \mathbb{R}^n$ turns the problem into a tractable *linear programming* problem.

If only some of the variables are constrained to be integers, while other variables are allowed to be non-integers, the problem is often referred to as *mixed-integer linear programming* (MILP) problem.

The algorithms for solving ILPs are, for example, the cutting plane methods, variants of the branch and bound methods, or a combination of both, such as branch and cut [13].

ILP solvers appear to be very fast at proving the optimality of a solution or estimating how far the solution is from the optimum. Currently the best proprietary ILP solvers are deemed Gurobi [14] and CPLEX [15].

1.3 Boolean Satisfiability and Satisfiability Modulo Theories

The problem of *Boolean satisfiability* (SAT) [16] is a problem of determining if there exists an interpretation that satisfies a given Boolean formula. Most SAT solvers today are still based on variations of the DPLL procedure [17]. However, significant advances in SAT solving techniques were made in the last decades thanks to better implementation techniques such as the *two-watched literal* approach for unit propagation [18] and conceptual improvements such as *conflict-driven clause learning* and *restarts* [19, 20].

This progress made SAT solvers applicable also to solving CSPs in general [21]. In the lazy clause generation approach [22], a Boolean formula corresponding to a finite domain CSP is lazily created during the computation. On the other hand, the motivation of gaining the advantages of techniques used both in SAT solvers and CP solvers leads to another hybrid approach, where some domain-specific reasoning is implemented within an SAT solver: SAT Modulo Theories (SMT) [23].

An SMT solver decides satisfiability of ground first-order logic formulae concerning some background theories, which is usually linear integer arithmetic in the case of scheduling applications. For example, a formula can contain clauses such as $v \vee w \vee (x - y \le 5)$, where v and w are Boolean variables and x and y are integer variables. During the computation, the theory solver checks whether the current assignment is feasible and possibly infers new facts that are then passed as clauses onto the SAT solver. Other background theories can be the theory of real numbers, the theory of integers, and the theories of various data structures such as lists, arrays, or bit vectors.

Just like CSP has been extended to COP, SMT can also include optimization, which is sometimes referred to as *Optimization Modulo Theories* (OMT). Optimization in SMT is formalized as follows. A weighted Max-SMT instance is an SMT instance where clauses may have an associated weight of falsification. Such clauses are called soft clauses, whereas the ones without weight are called hard clauses. A solution for this kind of instance must satisfy all hard clauses and minimize the sum of the weights of the falsified clauses.

In general, SAT and SMT solvers seem to be extremely fast at proving infeasibility and attaining an unsatisfiable core. On the other hand, when it comes to optimization, SMT solvers have the drawback that they either find an optimal solution or do not find any solution at all in the given time limit. The most efficient open-source SMT solvers are currently Z3 [24] and Yices [25].

1.4 Scheduling in IBM CP Optimizer

Recall that *scheduling* is a decision-making process where the goal is to allocate scarce resources to activities that require the resources to be performed. The allocation must be done in such a way that several constraints are satisfied to make the resulting schedule feasible. The most typical constraint is that the resources, whether machines or workers on the shop-floor, can perform one activity at a time, which means that the activities must not overlap in time. This requirement is referred to as *unary-resource* or *disjunctive* constraint.

In this thesis, we use mainly CP for modeling the problems, while the models are solved using IBM CP Optimizer. The primary motivation is that the CP Optimizer implements particular global constraints modeling unary resources and efficient filtering algorithms [12]. CP works with so-called *interval variables* whose start time and completion time are denoted by predicates StartOf and EndOf, and the difference between the completion time and the start time of the interval variable can be set using predicate LengthOf.

In addition, we use the concept of *optional interval variables* [26]. An optional interval variable can be set to be *present* or *absent*. The predicate PresenceOf is used to determine whether or not the interval variable is present in the resulting schedule. Whenever an optional interval variable is absent, it plays no role in most of the constraints, and predicates StartOf, EndOf, and LengthOf are set to 0.

The typical unary-resource constraints are modeled by the NoOverlap constraint. The NoOverlap constraint on a set of interval variables states that it constitutes a chain of non-overlapping interval variables, any interval variable in the chain being constrained to be completed before the start of the following interval variable in the chain. The interval variables that are absent do not affect the constraint.

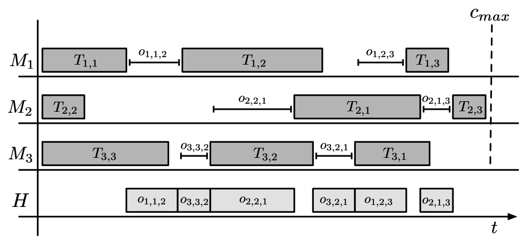
Other important constructs will be explained further in the text when needed.

2. Optional Activities for Non-overlapping Setups

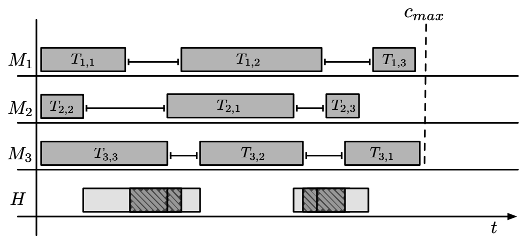
The problem studied in this chapter is inspired by the continuous production of plastic water tubes. In such productions, the factory brings in the material in the form of plastic granulate that is being in-house processed. The manufacturer has a stack of orders for manufacturing plastic tubes of various widths and lengths. The production has 13 machines that can produce different tubes in parallel. Different variants of tubes require different settings of the machines. Hence, when switching from one type of tube to another, a machine setter is required to visit the particular machine and make the tool adjustment. The goal is to process all orders as fast as possible.

As the tool adjustment is done by a single machine setter, he or she is likely to be the bottleneck of the production when the orders are not scheduled well. Given the assignment of the orders to the machines, the basic idea is to cluster similar tube variants next to each other, as these require little or no setup time to adjust the tool.

We model the problem as a scheduling problem where the tasks are dedicated to the machines and have sequence-dependent setup times. Each setup occupies an extra resource that is assumed to be unary, hence setups must not overlap in time. The goal is to minimize the makespan of the overall schedule. In this chapter, we design an Integer Linear Programming (ILP) model, five Constraint Programming (CP) models, and a heuristic algorithm.



(a) Feasible schedule.



(b) Infeasible schedule, setups are overlapping.

Figure 2.1: An illustration of a schedule with three machines and three tasks to be processed on each machine.

The main contributions of this chapter, which were presented in [1] and [2], are:

- formal definition of a new problem with non-overlapping setups
- exact approaches based on ILP and CP formalisms
- a very efficient hybrid heuristic yielding optimal or near-optimal schedules

2.1 Problem Specification

Informally speaking, the problem tackled in this chapter consists of a set of machines and a set of independent non-preemptive tasks, each of which is dedicated to one particular machine where it will be processed. Also, there are sequence-dependent setup times on each machine. In addition, these setups are to be performed by a human operator who is referred to as a machine setter. Such a machine setter cannot perform two or more setups at the same time. It follows that the setups on all the machines must not overlap in time. Examples of a feasible and infeasible schedule with 3 machines can be seen in Figure 2.1. Even though the schedule (Figure 2.1b) on the machines contains setup times, the schedule is infeasible since it would require overlaps in the schedule for the machine setter.

The aim is to find a schedule that minimizes the completion time of the latest task. It is clear that the latest task is on some machine and not in the schedule of a machine setter since the completion time of the last setup is followed by at least one task on a machine.

2.1.1 Related Work

There is a myriad of papers on scheduling with sequence-dependent setup times or costs [27], proposing exact approaches [28] as well as various heuristics [29]. But the research on the problems where the setups require extra resource is scarce.

An unrelated parallel machine problem with machine and job sequence-dependent setup times, studied by [30], considers also the non-renewable resources that are assigned to each setup, which affects the amount of time the setup needs and which is also included in the objective function. On the other hand, how many setups may be performed at the same time is disregarded. The authors propose a Mixed Integer Programming formulation along with some static and dynamic dispatching heuristics.

A lotsizing and scheduling problem with a common setup operator is tackled in [31]. The authors give ILP formulations for what they refer to as a dynamic capacitated multi-item multi-machine one-setup-operator lotsizing problem. Indeed, the setups to be performed by the setup operator are considered to be scheduled such that they do not overlap. However, these setups are not sequence-dependent in the usual sense. The setups are associated to a product whose production is to be commenced right after the setup and thus the setup time, i.e., the processing time of the setup, does not depend on a pair of tasks but only on the succeeding task.

A complex problem that involves machines requiring setups that are to be performed by operators of different capabilities has been addressed in [32]. The authors modeled the whole problem in the time-indexed formulation and solved it by decomposing the problem into smaller subproblems using Lagrangian Relaxation and solving the subproblems using dynamic programming. A feasible solution is then composed of the solutions to the subproblems by heuristics, and, if impossible, the Lagrangian multipliers are updated using the surrogate subgradient method as in [33]. The downside of this approach is that the time-indexed formulation yields a model of pseudo-polynomial size. This is not suitable for our problem as it poses large processing and setup times.

2.1.2 Formal Definition

Let $M = \{M_1, ..., M_m\}$ be a set of machines and for each $M_i \in M$, let $T^{(i)} = \{T_{i,1}, ..., T_{i,n_i}\}$ be a set of tasks that are to be processed on machine M_i, and let $T = \bigcup_{M_i \in M} T^{(i)} = \{T_{1,1}, ..., T_{m,n_m}\}$ denote the set of all tasks. Each task $T_{i,j} \in T$ is specified by its processing time $p_{i,j} \in \mathbb{N}$. Let $s_{i,j} \in \mathbb{N}_0$ and $C_{i,j} \in \mathbb{N}$ be start time and completion time, respectively, of task $T_{i,j} \in T$, which are to be found. All tasks are non-preemptive, hence, $s_{i,j} + p_{i,j} = C_{i,j}$ must hold.

Each machine $M_i \in M$ performs one task at a time. Moreover, the setup times between two consecutive tasks processed on machine $M_i \in M$ are given in matrix $O^{(i)} \in \mathbb{N}^{n_i \times n_i}$. That is, $o_{i,j,j'} = (O^{(i)})_{j,j'}$ determines the minimal time distance between the start time of task $T_{i,j'}$ and the completion time of task $T_{i,j}$ if task $T_{i,j'}$ is to be processed on machine M_i right after task $T_{i,j}$, i.e., $s_{i,j'} - C_{i,j} \geq o_{i,j,j'}$ must hold.

Let $H = \{h_1, \ldots, h_\ell\}$, where $\ell = \sum_{M_i \in M} n_i - 1$, be a set of setups that are to be performed by the machine setter. Each $h_k \in H$ corresponds to the setup of a pair of tasks that are scheduled to be processed in a row on some machine. Thus, function $st : H \to M \times T \times T$ is to be found. Also, $s_k \in \mathbb{N}_0$ and $C_k \in \mathbb{N}$ are start time and completion time of setup $h_k \in H$, which are to be found. Assuming $h_k \in H$ corresponds to the setup between tasks $T_{i,j} \in T$ and $T_{i,j'} \in T$, i.e., $st(h_k) = (M_i, T_{i,j}, T_{i,j'})$, it must hold that $s_k + o_{i,j,j'} = C_k$, also $C_{i,j} \leq s_k$, and $C_k \leq s_{i,j'}$. Finally, since the machine setter may perform at most one task at any time, it must hold that, for each $h_k, h_{k'} \in H, k \neq k'$, either $C_k \leq s_{k'}$ or $C_{k'} \leq s_k$.

The objective is to find such a schedule that minimizes the makespan, i.e., the latest completion time of any task:

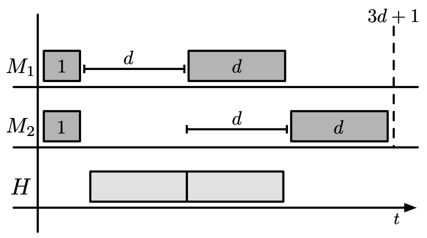
$$\min \max_{T_{i,j} \in T} C_{i,j} \tag{2.1}$$

2.1.3 Complexity

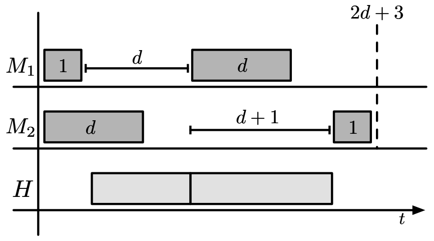
We note that minimizing the makespan for each machine separately does not guarantee a globally optimal solution. In fact, such a solution can be arbitrarily bad. Consider a problem depicted in Figure 2.2. It consists of two machines, M_1 and M_2, and two tasks on each machine, with processing times $p_{1,1} = p_{2,1} = 1, p_{1,2} = p_{2,2} = d$, where d is any constant greater than 2, and with setup times $o_{1,1,2} = o_{2,1,2} = d, o_{1,2,1} = o_{2,2,1} = d + 1$. Then, an optimal sequence on each machine yields a solution of makespan $3d + 1$, whereas choosing a suboptimal sequence on either of the machines gives optimal objective value $2d + 3$.

It is easy to see that the problem with sequence-dependent non-overlapping setups is strongly $\mathcal{NP}$-hard even for the case of one machine, i.e., $m = 1$, which can be shown by the reduction from the shortest Hamiltonian path problem.

Moreover, we showed in [2] that even the restricted problem where the task sequences on each machine are fixed is $\mathcal{NP}$-hard as well, suggesting another source of hardness. The proof is based on the observation that the problem with fixed sequences on machines is equivalent to $1|l_{ij} > 0, m\, n_1, \ldots, n_m\text{-chains}|C_{\max}$, i.e., a single machine scheduling problem with minimum time lags, where the precedence



(a) A problem instance where optimal sequences on machines lead to a sub-optimal solution.



(b) Sub-optimal sequence on one machine yields a globally optimal solution.

Figure 2.2: Solving the problem greedily for each machine separately can lead to arbitrarily bad solutions. The numbers depict the processing times of the tasks and setups.

graph has a form of m chains of lengths $n_1, \dots, n_m$, followed by a single common task. But this problem even with two chains, i.e., $1|l_{ij} > 0, m\,2\text{-chains}|C_{\max}$, is known to be strongly $\mathcal{NP}$-hard by the reduction from 3-PARTITION problem [34].

In the following sections, we propose two exact approaches.

2.2 Integer Linear Programming Model

Antonín Novák proposed a formulation that models the problem with two parts. The first part handles the scheduling of tasks on the machines using efficient *rank-based model* [35]. This approach uses binary variables $x_{i,j,q}$ to encode whether task $T_{i,j} \in T^{(i)}$ is scheduled on q-th position in the permutation on machine $M_i \in M$. Another variable is $\tau_{i,q}$ denoting the start time of a task that is scheduled on q-th position in the permutation on machine $M_i \in M$.

The second part of the model resolves the question, in which order and when the setups are performed by a machine setter. There, we need to schedule all setups H, where the setup time π_k of the setup $h_k \in H$ is given by the corresponding pair of tasks on the machine.

Let us denote the set of all natural numbers up to n as $[n] = \{1, \dots, n\}$. We define the following function $\phi : H \to M \times [\max_{M_i \in M} n_i]$ (e.g., $\phi(h_k) = (M_i, q)$), that maps $h_k \in H$ to setups between the tasks scheduled at positions q and $q+1$ on machine $M_i \in M$. Since the time of such setup is a variable (i.e., it depends on the pair of consecutive tasks on M_i), rank-based model would not be linear. Therefore, we use the *relative-order* (also known as disjunctive) model [36, 37] that admits processing time given as a variable. Its disadvantage over the rank-based model is that it introduces a *big M* constant in the constraints, whereas the rank-based model does not. See Figure 2.3 for meaning of the variables.

The full model is stated as:

$$\min \quad c_{max} \tag{2.2}$$

s.t.

$$c_{max} \geq \tau_{i,n_i} + \sum_{T_{i,j} \in T^{(i)}} p_{i,j} \cdot x_{i,j,n_i} \quad \forall M_i \in M \tag{2.3}$$

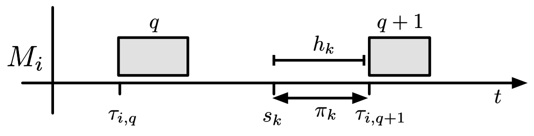


Figure 2.3: Meaning of the variables in the model.

$$\sum_{q \in [n_i]} x_{i,j,q} = 1 \quad \forall M_i \in M, \forall T_{i,j} \in T^{(i)} \tag{2.4}$$

$$\sum_{T_{i,j} \in T^{(i)}} x_{i,j,q} = 1 \quad \forall M_i \in M, \forall q \in [n_i] \tag{2.5}$$

$$s_k + \pi_k \le s_l + \mathcal{M} \cdot (1 - z_{k,l}) \quad \forall h_l, h_k \in H : l < k \tag{2.6}$$

$$s_l + \pi_l \le s_k + \mathcal{M} \cdot z_{k,l} \quad \forall h_l, h_k \in H : l < k \tag{2.7}$$

$$\pi_k \ge o_{i,j,j'} \cdot (x_{i,j,q} + x_{i,j',q+1} - 1)$$
$$\forall h_k \in H : \phi(h_k) = (M_i, q), \forall T_{i,j}, T_{i,j'} \in T^{(i)} \tag{2.8}$$

$$s_k + \pi_k \le \tau_{i,q+1} \quad \forall h_k \in H : \phi(h_k) = (M_i, q) \tag{2.9}$$

$$s_k \ge \tau_{i,q} + \sum_{T_{i,j} \in T^{(i)}} p_{i,j} \cdot x_{i,j,q} \quad \forall h_k \in H : \phi(h_k) = (M_i, q) \tag{2.10}$$

where

$$c_{max} \in \mathbb{R}_0^+ \tag{2.11}$$

$$\tau_{i,q} \in \mathbb{R}_0^+ \quad \forall M_i \in M, \forall q \in [n_i] \tag{2.12}$$

$$s_k, \pi_k \in \mathbb{R}_0^+ \quad \forall h_k \in H \tag{2.13}$$

$$x_{i,j,q} \in \{0, 1\} \quad \forall M_i \in M, \forall T_{i,j} \in T^{(i)}, \forall q \in [n_i] \tag{2.14}$$

$$z_{k,l} \in \{0, 1\} \quad \forall h_k, h_l \in H : l < k \tag{2.15}$$

The constraint (2.3) computes makespan of the schedule while constraints (2.4)–(2.5) states that each task occupies exactly one position in the permutation and that each position is occupied by exactly one task. Constraints (2.6) and (2.7) guarantee that setups do not overlap. $\mathcal{M}$ is a constant that can be set as $|H| \cdot \max_{i,j,j'} o_{i,j,j'}$. Constraint (2.8) sets processing time π_k of the setup $h_k \in H$ to $o_{i,j,j'}$ if task $T_{i,j'}$ is scheduled on machine M_i right after task $T_{i,j}$. Constraints (2.9) and (2.10) are used to avoid conflicts on machines. The constraint (2.9) states that a task cannot start earlier than its preceding setup finishes. Similarly, the constraint (2.10) states that setup is scheduled after the corresponding task on the machine finishes.

Notice that we can reduce the number of variables in the model due to the structure of the problem. We fix values of some of the $z_{k,l}$ variables according to the following rule. Let $h_k, h_l \in H$ such that $\phi(h_k) = (M_i, q)$ and $\phi(h_l) = (M_i, v)$ for any $M_i \in M$. Then, $q < v \Rightarrow z_{k,l} = 1$ holds in some optimal solution. Note that the rule holds only for setups following from the same machine.

The rule states that the relative order of setups on the same machine is determined by the natural ordering of task positions on that machine. See for example setups $o_{1,1,2}$ and $o_{1,2,3}$ in Figure 2.1. Since these setups follow from the same machine, their relative order is already predetermined by positions of the respective tasks. We note that the presolve of the solver was not able to deduce these rules on its own.

2.2.1 Formulation for a Single Machine

The problem with a single machine ($M_i \in M$) reduces to the shortest Hamiltonian path in the graph defined by setup time matrix $O^{(i)}$. To solve this problem, we transform it to the Traveling Salesperson Problem by introducing a dummy task $T_{i,0} \in T^{(i)\prime} = T^{(i)} \cup \{T_{i,0}\}$ that has zero setup times with all other tasks, i.e., $o_{i,0,j} = o_{i,j,0} = 0$, $\forall T_{i,j} \in T^{(i)\prime}$. Then, we use a well-known sub-tour elimination [38, 39] ILP model to solve it:

$$\min \sum_{T_{i,j} \in T^{(i)\prime}} \sum_{T_{i,j'} \in T^{(i)\prime}} o_{i,j,j'} \cdot y_{j,j'} + \sum_{T_{i,j} \in T^{(i)\prime}} p_{i,j} \tag{2.16}$$

s.t.

$$\sum_{T_{i,j} \in T^{(i)\prime}} y_{j,j'} = 1 \quad \forall T_{i,j'} \in T^{(i)\prime} \tag{2.17}$$

$$\sum_{T_{i,j'} \in T^{(i)\prime}} y_{j,j'} = 1 \quad \forall T_{i,j} \in T^{(i)\prime} \tag{2.18}$$

$$\sum_{T_{i,j}, T_{i,j'} \in S} y_{j,j'} \leq |S| - 1 \quad \forall S \subset T^{(i)\prime} \tag{2.19}$$

where

$$y_{j,j'} \in \{0, 1\} \quad \forall T_{i,j}, T_{i,j'} \in T^{(i)\prime} \tag{2.20}$$

The variable $y_{j,j'}$ indicates whether task $T_{i,j}$ is immediately followed by task $T_{i,j'}$. We solve the model in a lazy way, i.e., without constraints (2.19), that are lazily generated during the solution by a depth-first search algorithm. Note that the machine setter does not need to be modeled for the single machine problem.

2.2.2 Additional Improvements

We use the following improvements of the model that have a positive effect on the solver performance.

1. **Warm Starts.** The solver is supplied with an initial solution. It solves a relaxed problem, where it relaxes on the condition that setups do not overlap. Such a solution is obtained by solving the shortest Hamiltonian path problem given by setup time matrix $O^{(i)}$ independently for each machine $M_i \in M$, as described in Section 2.2.1. Since such a solution might be infeasible for the original problem, we transform it in a polynomial time into a feasible one. It is done in the following way. For each setup among all machines, we set the start time of k-th setup on machine M_i, $i \geq 2$, to the completion time of k-th setup on machine M_{i-1}. For the setups on machine M_1, the start time of $(k+1)$-th setup is set to the completion time of k-th setup on machine M_m.

2. **Lower Bounds.** We supply a lower bound on c_{max} variable given as the maximum of all best proven lower bounds of model (2.16)–(2.20) among all machines $M_i \in M$ (see Section 2.2.1).

2.3 Constraint Programming Models

Another way how the problem at hand can be tackled is to use the modeling approach based on the Constraint Programming (CP) formalism, which is introduced in Sections 1.1 and 1.4.

The CP models are constructed as follows. We introduce interval variables $I_{i,j}$ for each $T_{i,j} \in T$, and the lengths of these interval variables are set to the corresponding processing times:

$$\text{LengthOf}(I_{i,j}) = p_{i,j} \tag{2.21}$$

The sequence is resolved using the NoOverlap constraint (see Section 1.4). In addition, the NoOverlap(I, $O^{(i)}$) constraint is given a so-called *transition distance* matrix $O^{(i)}$, which expresses a minimal delay that must elapse between two successive interval variables. More precisely, if $I_{i,j}, I_{i,j'} \in I$, then $(O^{(i)})_{j,j'}$ gives a minimal allowed time difference between StartOf($I_{j'}$) and EndOf(I_j). Hence, the following constraint is imposed, $\forall M_i \in M$:

$$\text{NoOverlap}\Big(\bigcup_{T_{i,j} \in T^{(i)}} \{I_{i,j}\}\ ,\ O^{(i)} \Big) \tag{2.22}$$

The objective function is to minimize the makespan:

$$\min \max_{T_{i,j} \in T} \text{EndOf}(I_{i,j}) \tag{2.23}$$

This model would already solve the problem if the setups were not required to be non-overlapping. In what follows we describe five ways how the non-overlapping setups are resolved. Constraints (2.21)–(2.23) are part of each of the following model.

2.3.1 CP1: with Implications

Let us introduce $I^{st}_{i,j}$ for each $T_{i,j} \in T$ representing the setup after task $T_{i,j}$. There is $\sum_{M_i \in M} n_i$ such variables. As the interval variable $I^{st}_{i,j}$ represents the setup after task $T_{i,j}$, we use the constraint EndBeforeStart(I_1, I_2), which ensures that interval variable I_1 is completed before interval variable I_2 can start. Thus, the following constraint needs to be added, $\forall M_i \in M, \forall T_{i,j} \in T^{(i)}$:

$$\text{EndBeforeStart}(I_{i,j}, I^{st}_{i,j}) \tag{2.24}$$

To ensure that the setups do not overlap in time is enforced through the following constraint:

$$\text{NoOverlap}\Big(\bigcup_{T_{i,j} \in T} \{I^{st}_{i,j}\} \Big) \tag{2.25}$$

Notice that this constraint is unique and it is imposed over all the interval variables representing setups on all machines. This NoOverlap constraint does not need any transition distance matrix as the default values 0 are desired.

Since it is not known a priori which task will follow task $T_{i,j}$, the quadratic number of implications determining the precedences and lengths of the setups must be imposed. For this purpose, the predicate TypeOfNext is used.

TypeOfNext(I) gives the *type* of the interval variable that is to be processed in the chain right after interval variable I. In our case, since all tasks are in general of different types, TypeOfNext$(I_{i,j})$ gives the index j' of the task corresponding to the interval variable $I_{i,j'}$ that is succeeding $I_{i,j}$. Thus, the following constraints are added, $\forall M_i \in M, \forall T_{i,j}, T_{i,j'} \in T^{(i)}, j \neq j'$:

$$\text{TypeOfNext}(I_{i,j}) = j' \Rightarrow \text{EndOf}(I^{st}_{i,j}) \leq \text{StartOf}(I_{i,j'}) \tag{2.26}$$

$$\text{TypeOfNext}(I_{i,j}) = j' \Rightarrow \text{LengthOf}(I^{st}_{i,j}) = o_{i,j,j'} \tag{2.27}$$

Note that the special value when an interval variable is the last one in the chain is used to turn the last setup on a machine into a dummy one.

2.3.2 CP2: with Element Constraints

Setting the lengths of the setups can be substituted by the element constraint, which might be beneficial as global constraints are usually more efficient. More precisely, this model contains also constraints (2.24), (2.25), and (2.26), but constraint (2.27) is substituted as follows.

Assume the construct Element$(Array, k)$ returns the k-th element of $Array$, $(O^{(i)})_j$ is the j-th row of matrix $O^{(i)}$, and TypeOfNext$(I_{i,j})$ again returns the index of the interval variable that is to be processed right after $I_{i,j}$. Then the following constraint is added, for each $I^{st}_{i,j}$:

$$\text{LengthOf}(I^{st}_{i,j}) = \text{Element}\left((O^{(i)})_j, \text{TypeOfNext}(I_{i,j})\right) \tag{2.28}$$

2.3.3 CP3: with Optional Interval Variables

In this model, we use the concept of *optional interval variables* (see Section 1.4). We introduce optional interval variable $I^{opt}_{i,j,j'}$ for each pair of distinct tasks on the same machine, i.e., $\forall M_i \in M, \forall T_{i,j}, T_{i,j'} \in T^{(i)}, j \neq j'$. There are $\sum_{M_i \in M} n_i(n_i - 1)$ such variables. The lengths of these interval variables are set to corresponding setup times:

$$\text{LengthOf}(I^{opt}_{i,j,j'}) = o_{i,j,j'} \tag{2.29}$$

To ensure that the machine setter does not perform more than one task at the same time, the following constraint is added:

$$\text{NoOverlap}\Big(\bigcup_{\substack{T_{i,j}, T_{i,j'} \in T \\ j \neq j'}} \{I^{opt}_{i,j,j'}\}\Big) \tag{2.30}$$

In this case, to ensure that the setups are indeed processed in between two consecutive tasks, we use the constraint EndBeforeStart(I_1, I_2), which ensures that interval variable I_1 is completed before interval variable I_2 can start, but if either of the interval variables is absent, the constraint is implicitly satisfied. Thus, the following constraints are added, $\forall I^{opt}_{i,j,j'}$:

$$\text{EndBeforeStart}(I_{i,j}, I^{opt}_{i,j,j'}) \tag{2.31}$$

$$\text{EndBeforeStart}(I^{opt}_{i,j,j'}, I_{i,j'}) \tag{2.32}$$

Finally, in order to ensure the correct presence of optional interval variables, the predicate PresenceOf is used. Thus, the following constraint is imposed, $\forall I^{opt}_{i,j,j'}$:

$$\text{PresenceOf}(I^{opt}_{i,j,j'}) \Leftrightarrow \text{TypeOfNext}(I_{i,j}) = j' \tag{2.33}$$

Notice that each $I_{i,j}$ (except for the last one on a machine) is followed by exactly one setup. Thus, we tried using a special constraint called Alternative, which ensures that exactly one interval variable from a set of variables is present. However, preliminary experiments showed that adding this constraint is counter-productive.

2.3.4 CP4: with Cumulative Function

In this model, the machine setter is represented as a cumulative function to avoid the quadratic number of constraints of CP1 and CP2 or the quadratic number of optional task variables of CP3. The definition of the non-negative cumulative function has a linear number of terms.

Again, we use interval variables $I_{i,j}$ for each $T_{i,j} \in T$ to model the execution of each task $T_{i,j} \in T$ and $I^{st}_{i,j}$ for each $T_{i,j} \in T$ representing the setup after task $T_{i,j}$. See Figure 2.4 for the illustration of main concepts.

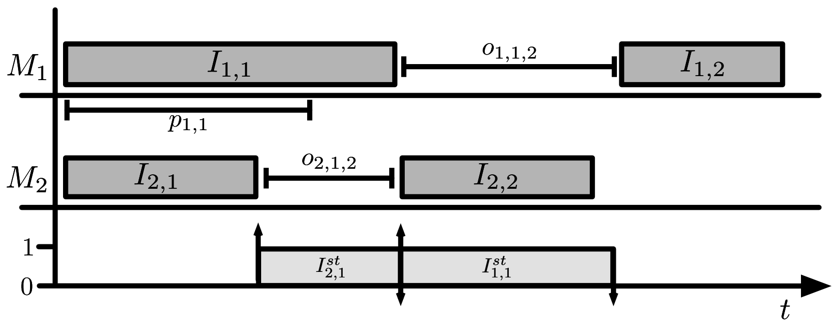


Figure 2.4: Illustration of variables and constraints for CP4. The length of $I_{1,1}$ is greater than $p_{1,1}$ as the completion of $I_{1,1}$ must wait for the machine setter to complete the setup $I^{st}_{2,1}$.

The idea now is that the lengths of the interval variables are not fixed. The length of interval variables $I_{i,j}$ is increased to wait for the machine setter if he or she is busy on another machine, i.e., the length of interval variable $I_{i,j}$ is at least the processing time $p_{i,j}$ but may be prolonged until the machines setter is available. Thus, the constraints (2.21) on the lengths of the interval variables $I_{i,j}$ are relaxed because they must only be greater than the corresponding processing times:

$$\text{LengthOf}(I_{i,j}) \geq p_{i,j} \tag{2.34}$$

The length of the setup executed by the machine setter will be determined by the corresponding tasks. In particular, as the last setup on a machine becomes a dummy setup (of zero length), we merely set the length of the setup variables to be at least zero:

$$\text{LengthOf}(I^{st}_{i,j}) \geq 0 \tag{2.35}$$

The cumulative function is built thanks to primitive $\text{Pulse}(a, h)$ that specifies that h unit of resource is used during interval a. The cumulative function is composed of Pulse terms for each $I^{st}_{i,j}$ representing the usage of a machine setter, and the cumulative function must remain lower than 1 because there is a single machine setter:

$$\sum_{T_{i,j} \in T} \text{Pulse}(I^{st}_{i,j}, 1) \leq 1 \tag{2.36}$$

What remains to be done is to synchronize start and completion times between the tasks and setups. This is done using the constraint $\text{EndAtStart}(I_1, I_2)$, which ensures that interval variable I_1 is completed exactly when interval variable I_2 starts, and by $\text{StartOfNext}(I_{i,j})$, which gives the start time of the interval variable that is to be processed right after $I_{i,j}$. Thus, the following constraints are added, for each $I^{st}_{i,j}$:

$$\text{EndAtStart}(I_{i,j}, I^{st}_{i,j}) \tag{2.37}$$

$$\text{EndOf}(I^{st}_{i,j}) \geq \text{StartOfNext}(I_{i,j}) \tag{2.38}$$

Note that the inequality in constraint (2.38) is necessary because StartOfNext gives 0 for the last task on a machine and thus the completion time of the last setup is equal to its start time (hence the length of the setup is 0). Also, note that the setups cannot be shorter than required due to constraint (2.22).

2.3.5 CP5: without Setup Variables

This model exploits the same idea as CP4, but we can go even further and completely omit the interval variables representing the setups. In this model, the interval variables $I_{i,j}$ are, again, relaxed because they must only be greater than the corresponding processing times, i.e., constraint (2.34) is kept.

The main difference in this model is that the cumulative function only requires the introduction of interval variables S_i, one for each machine M_i. See Figure 2.5 for the illustration of main concepts. Variable S_i starts with the first task of the machine M_i and ends with the last task, which is enforced by the Span constraint:

$$\text{Span}\Big(S_i, \bigcup_{T_{i,j} \in T^{(i)}} \{I_{i,j}\}\Big) \tag{2.39}$$

The cumulative function is now realized as follows:

$$\sum_{M_i \in M} \text{Pulse}(S_i, 1) - \sum_{T_{i,j} \in T} \text{Pulse}(I_{i,j}, 1) \leq 1 \tag{2.40}$$

The first term enforces that the cumulative function remains non-negative when the first task $T_{i,j}$ of the machine M_i starts and that the machine setter is not required at the end of the last task $T_{i,j}$ of the machine M_i. The second term enforces that the machine setter is available when a task $T_{i,j}$ ends, possibly after a waiting time as allowed by constraints (2.34), and that the machine setter becomes available again when a task $T_{i,j}$ starts.

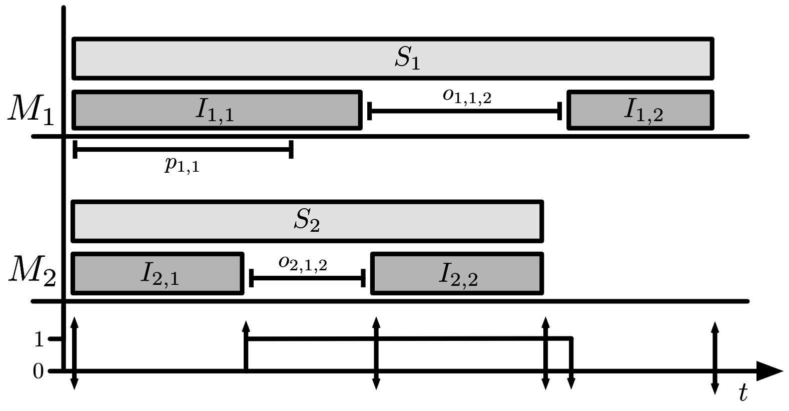


Figure 2.5: Illustration of variables and constraints for CP5.

Despite the lowest number of variables, the main drawback of this model is that the schedule of the machine setter becomes implicit which leads to less filtering.

2.3.6 Additional Improvements

We use the following improvements:

1. **Search Phases.** Automatic search in the solver is well tuned-up for most types of problems, leveraging the newest knowledge about variable selection and value ordering heuristics. In our case, however, preliminary results showed that the solver struggles to find any feasible solution already for small instances. It is clear that it is easy to find some feasible solution, e.g., by setting an arbitrary order of tasks on machines and then shifting the tasks to the right such that the setups do not overlap. To make the solver find some feasible solution more quickly, we set the search phases such that the sequences on machines are resolved first, and then the sequences of setups for the machine setter are resolved. This is included in all the CP models described.

2. **Warm Starts.** Similarly to improvement (1) in Section 2.2.2, we boost the performance by providing the solver with a starting point. We do this only for CP3 as the preliminary numerical experiments showed a slight superiority of CP3.

   More precisely, we first find an optimal sequence of tasks minimizing makespan on each machine separately and then we set those interval variables $I^{opt}_{i,j,j'}$ to be present if $T_{i,j'}$ is sequenced directly after $T_{i,j}$ on machine M_i. This is all that we set as the starting point. Notice that unlike in Section 2.2.2, we do not calculate the complete solution but we let the solver do it. The solver then quickly completes the assignment of all the variables such that it gets a solution of reasonably good objective value.

   Note that the optimal sequences on machines are solved using ILP so it can be seen as a hybrid approach. This model with warm starts is in what follows referred to as *CP3ws*.

2.4 Heuristic Approach

We propose an approach that guides the solver quickly towards solutions of very good quality but cannot guarantee the optimality of what is found. There are two main phases of this approach. In the first phase, the model is decomposed such that its subproblems are solved optimally or near-optimally, and then the solutions of the subproblems are put together so as to make a correct solution of the whole problem. In the second phase, the solution found is locally improved by repeatedly adjusting the solution in promising areas. More details follow.

2.4.1 Decomposition Phase

The idea of the model decomposition is as follows. First, again, we find an optimal sequence of tasks minimizing makespan on each machine separately, as described in Section 2.2.1. Second, given these sequences on each machine, the setups to be performed are known, hence, the lengths of the setups are fixed as well as the precedence constraints with respect to the tasks on machines. Thus, all that needs to be resolved is the order of setups.

The pseudocode is given in Algorithm 1. It takes one machine at a time and finds an optimal sequence for it minimizing the makespan. The time limit for the computation of one sequence on a machine is given in such a way that there is a proportional remaining time limit for the rest of the algorithm. OptimalSeq(i, $TimeLimit$) returns the best sequence it finds on machine $M_i \in M$ in the given $TimeLimit$. The $TimeLimit$ is computed using $RemainingTime()$, which is the time limit for the entire run of the algorithm minus the time that already elapsed from the beginning of the run of the algorithm. In the end, the solution is found using the knowledge of the sequences on each machine $M_i \in M$.

Algorithm 1 Solving the decomposed model.

```
function SolveDecomposed
    for each $M_i \in M$ do
        $TimeLimit \leftarrow RemainingTime()/(m - i + 2)$
        $Seq_i \leftarrow$ OptimalSeq($i$, $TimeLimit$)
    end for
    Return Solve($Seq$, $RemainingTime()$)
end function
```

2.4.2 Improving Phase

Once we have some solution to the problem, the idea of the heuristic is to improve it by applying the techniques known as local search [40] and large neighborhood search [41].

It is clear that in order to improve the solution, something needs to be changed on the *critical path*, which is such a sequence of setups and tasks on machines that the completion time of the last task equals the makespan and that none of these tasks and setups can be shifted to the left without violating resource constraints (see an example in Figure 2.6). Hence, we find the critical path first.

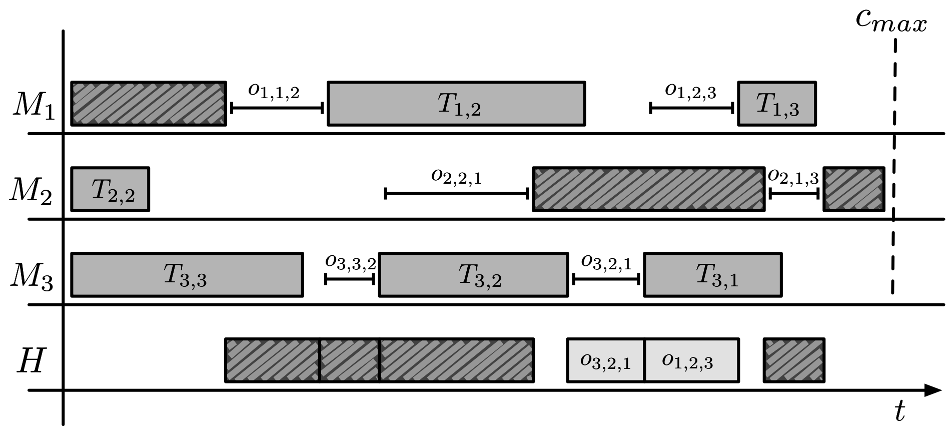


Figure 2.6: An illustration of the critical path depicted by dashed rectangles.

The most promising place to be changed on the critical path could be the longest setup. Hence, we find the longest setup on the critical path, then we prohibit the two consecutive tasks corresponding to the setup from being processed in a row again and re-optimize the sequence on the machine in question. Two tasks are precluded from following each other by setting the corresponding setup time to an infinite value. Also, we add extra constraint restricting the makespan to be less than the incumbent best objective value found. The makespan on one machine being equal to or greater than the incumbent best objective value found cannot lead to a better solution.

After a new sequence is found, the solution to the whole problem is again re-optimized subject to the new sequence. The algorithm continues this way until the sequence re-optimization returns infeasible, which happens due to the extra constraint restricting the makespan. It means that the solution quality deteriorated too much and it is unlikely to find a better solution locally at this state. Thus, the algorithm reverts to the initial solution obtained from the decomposed model, restores the original setup times matrices, and tries to prohibit another setup time on the critical path. For this purpose, the list of *nogoods* to be tried is computed once from the first critical path, which is just a list of setups on the critical path sorted in non-increasing order of their lengths. The whole iterative process is repeated until the total time limit is exceeded or all the nogoods are tried.

The entire heuristic algorithm is hereafter referred to as LOFAS (Local Optimization for Avoided Setup). The pseudocode is given in Algorithm 2.

Preliminary experiments confirmed the well-known facts that ILP using the lazy approach is very efficient for searching an optimal sequence on one resource and that CP is more efficient for minimizing makespan when the lengths of interval variables and the precedences are fixed. Nevertheless, for instances with many tasks, the solution involving ILP might be computationally infeasible. Hence, we also propose to find a suboptimal sequence for each machine by a heuristic method. Thus, in what follows, we distinguish the following two variants of the algorithm

1. **Exact subproblem.** The sequence is found by ILP with lazy subtour elimination, as described in 2.2.1. In the experiments below, we denote this

Algorithm 2 Local Optimization for Avoided Setup

```
function LOFAS
    $S^{init} \leftarrow$ SolveDecomposed
    $S^{best} \leftarrow S^{init}$
    $P_{crit} \leftarrow$ critical path in $S^{init}$
    $nogoods \leftarrow H \cap P_{crit}$
    sort $nogoods$ in non-increasing order of lengths
    for each $h_k \in nogoods$ do
        $h_{k'} \leftarrow h_k$
        while true do
            $(M_i, T_{i,j}, T_{i,j'}) \leftarrow st(h_{k'})$
            $o_{i,j,j'} \leftarrow \infty$
            add: $\max_{T_{i,j} \in T^{(i)}} C_{i,j} < ObjVal(S^{best})$
            $TimeLimit \leftarrow RemainingTime()/2$
            $Seq_i \leftarrow$ OptimalSeq($i$, $TimeLimit$)
            if $Seq_i$ is $infeasible$ then
                Revert to $S^{init}$
                Restore original $O^{(i)}, \forall M_i \in M$
                break
            end if
            $S^{new} \leftarrow$ Solve($Seq$, $RemainingTime()$)
            if $ObjVal(S^{best}) > ObjVal(S^{new})$ then
                $S^{best} \leftarrow S^{new}$
            end if
            if $RemainingTime() \leq 0$ then
                return $S^{best}$
            end if
            $P_{crit} \leftarrow$ critical path in $S^{new}$
            $h_{k'} \leftarrow$ longest setup $\in H \cap P_{crit}$
        end while
    end for
    return $S^{best}$
end function
```

variant as *LOFAS*.

2. **Heuristic subproblem.** The suboptimal sequence is found by *Guided Local Search* algorithm implemented in Google's *OR-Tools* [42]. Note that this algorithm is a metaheuristic, hence, it consumes all the time limit assigned even if the objective value is not improving for several iterations (i.e., cannot prove optimality). In the experiments, we denote this variant as *LOFAS /w heur.*

2.5 Experiments

For the implementation of the CP approaches, we used the IBM CP Optimizer version 12.9 [11]. The only parameter that we adjusted is `Workers`, which is the number of threads the solver can use and which we set to 1. In Google's *OR-Tools*, we only set `LocalSearchMetaheuristic` to `GuidedLocalSearch`.

For the ILP approach, we used Gurobi solver version 8.1 [14]. The parameters that we adjust are `Threads`, which we set to 1, and `MIPFocus`, which we set to 1 in order to make the solver focus more on finding solutions of better quality rather than proving optimality. We note that tuning the parameters with Gurobi Tuning Tool did not produce better values over the baseline ones.

The experiments were run on a Dell PC with an Intel® Core™ i7-4610M processor running at 3.00 GHz with 16 GB of RAM. We used a time limit of 60 seconds per problem instance.

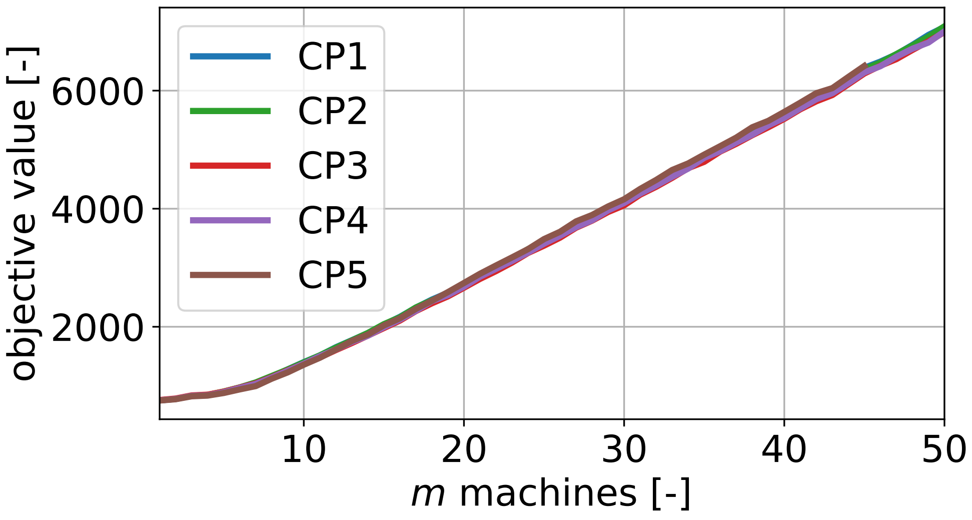
2.5.1 Problem Instances

We evaluated the approaches on randomly generated instances of various sizes with the number of machines m ranging from 1 to 50 and the number of tasks on each machine $n_i = n$, $\forall M_i \in M$, ranging from 2 to 50. Thus, we generated $50 \times 49 = 2450$ instances in total. Processing times of all the tasks and setup times are chosen uniformly at random from the interval $[1, 50]$. Instances are publicly available at `https://github.com/CTU-IIG/NonOverlappingSetupsScheduling`.

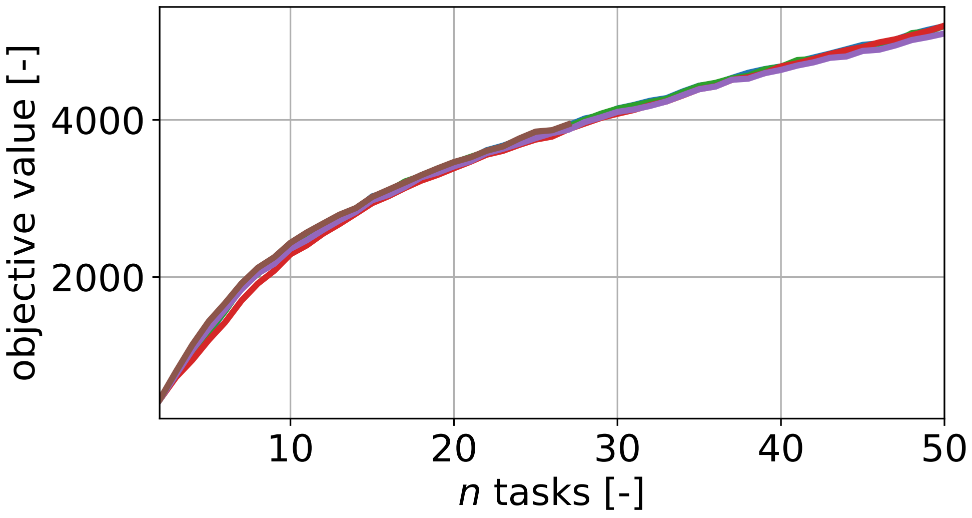
2.5.2 Scalability Results

Figure 2.7a shows the dependence of the best objective value found by CP models within the 60s time limit on the number of machines, averaged over the various number of tasks. Analogically, Figure 2.7b shows the dependence of the best objective value on the number of tasks, averaged over the varying number of machines.

The results show that the performances of CP models are almost equal (the graphs almost amalgamate). However, it can be seen that the curve of CP5 is not complete. It is because CP5 fails to find any solution for 88 of the largest instances, despite having the lowest number of variables. We note that CP3 is the best on average but the advantage is not significant. On the other hand, CP4 has a better worst-case performance. We will no longer distinguish between the CP models and we will use the minimum of all five CP models that will be referred to as *CPmin*.

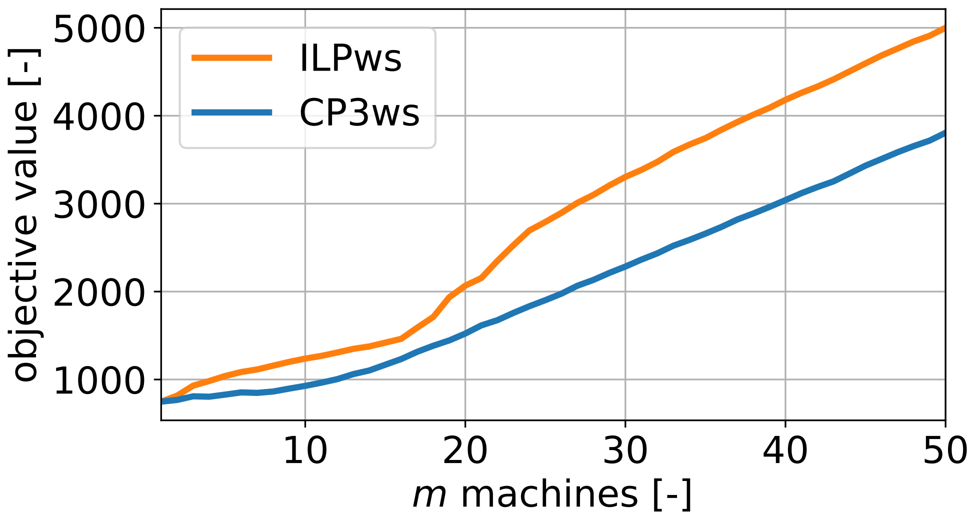


(a) Mean objective value for different number of machines m.

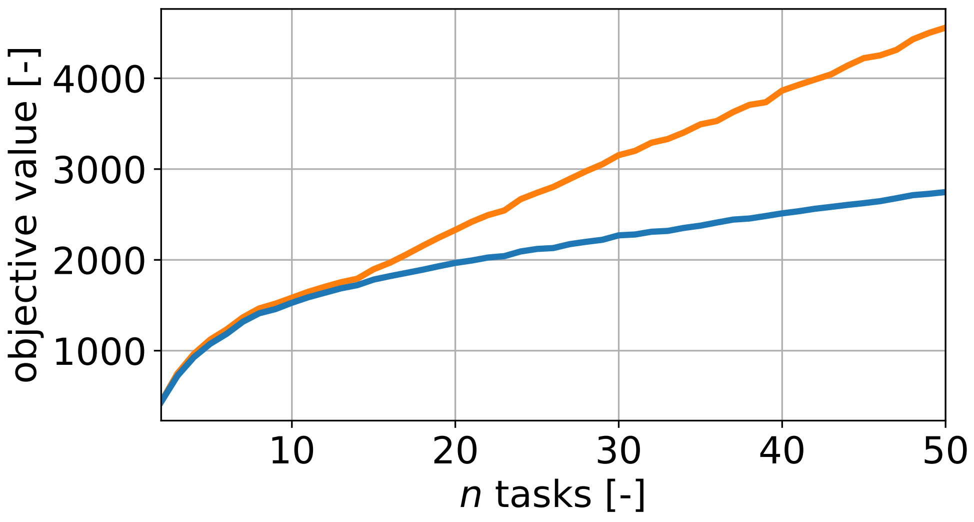


(b) Mean objective value for different number of tasks n.

Figure 2.7: Comparison of CP models.



(a) Mean objective value for different number of machines m.



(b) Mean objective value for different number of tasks n.

Figure 2.8: Comparison of exact models with warm starts.

The comparison of CP3ws to ILPws (models with warm starts) is shown in Figure 2.8. Recall that both the approaches get a warm start in a certain sense. The results confirm the lower performance of the ILP approach. When the ILP model did not get the initial solution as a warm start, it was not able to find any solution even for very small instances (i.e., 2 machines and 8 tasks). In fact, the objective value found by the ILPws is often the objective value of the greedy initial solution given as the warm start (as described in Section 2.2.2).

Further, we compare the best objective value found by the heuristic algorithm LOFAS and LOFAS */w heur.* from Section 2.4 against CPmin and ILPws. The results are shown in Figure 2.9. Note that we omit the results of CP3ws in Figure 2.9 as the results were almost the same as those of LOFAS and the curves amalgamated.

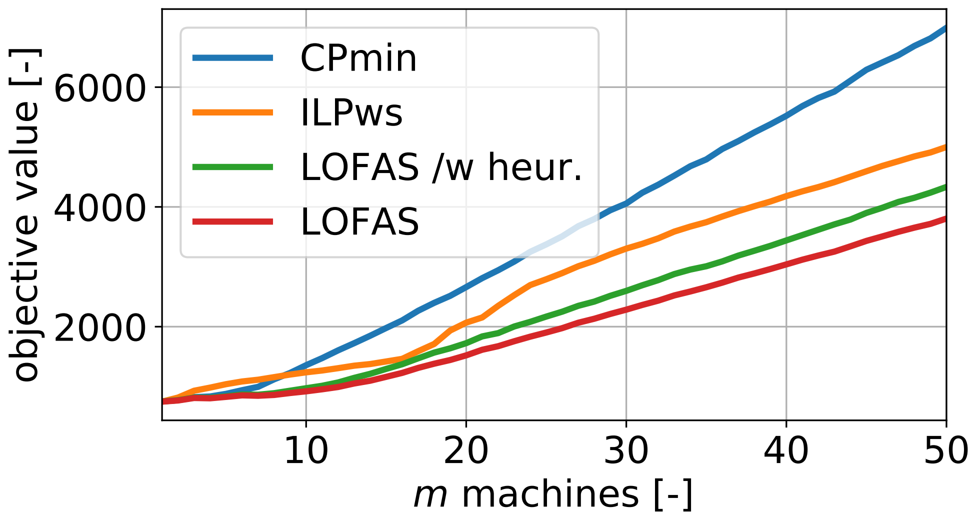
To obtain better insight into the performance of the proposed methods, we compared the resulting distributions of achieved objectives from each method. We took the results of each method for all instances and ordered them in a non-decreasing way with respect to achieved objective value and plotted them. The results are displayed in Figure 2.10. It can be seen that the proposed heuristics are able to find the same or better solutions in nearly all cases. Furthermore, we note that LOFAS */w heur.* outperformed both CPmin and ILPws as well. For the ILPws, one can notice a spike at around 65 % of instances. This is caused by the fact that for some instances, the ILP solver was not able to improve upon the initial warm start solution in the given time limit and these instances thus contribute to the distribution with larger objective values.

A comparison of LOFAS to CP3ws on larger instances [1] showed a slight superiority of LOFAS. However, LOFAS still did not find any solution to the biggest instances because the time limit was exceeded during the decomposition phase, i.e., during seeking an optimal sequence for a machine. This was the motivation for developing LOFAS */w heur.*

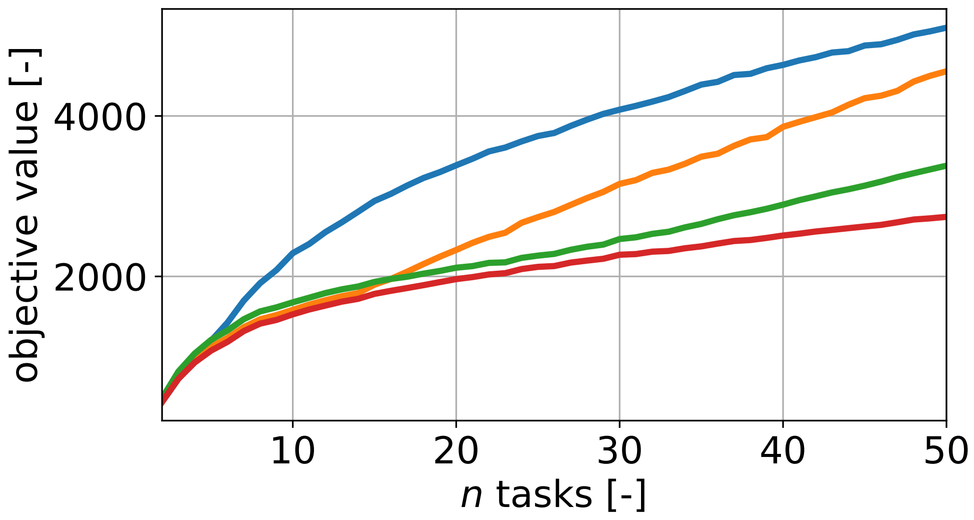
2.5.3 Effect of Exact/Heuristic Subproblem Solution

In this section, we compare solution quality produced by LOFAS heuristics with different methods for solving the machine subproblem, i.e., the method $\textsc{Seq}(i, \mathit{TimeLimit})$. The experiments were designed to assess if and how much different objective values are achieved when using the heuristic solution of the subproblem and if the heuristic variant scales better.

We have generated instances with $m \in \{5, 10, 15, 20\}$ machines, the number of tasks on each machine n ranging from 50 up to 1000. For each combination of m and n, we have generated 10 instances. The results are reported in Table 2.1. The column *objective* denotes the mean objective value if all instances were solved in the given time limit. Otherwise, we report the number of instances that were solved within the time limit. The results confirm the hypothesis that LOFAS gives better solutions regarding the objective, whereas LOFAS */w heur.* is able to find some solutions for larger instances when LOFAS does not manage to find any solution. More precisely, LOFAS scales only up to 300 tasks on 15 machines or 350 tasks on 5 machines. A heuristic solution of the subproblem in LOFAS */w heur.* allows obtaining a solution for instances of up to 1000 tasks on 5 machines. However, with the increasing number of machines, the CP solver struggles



(a) Mean objective value for different number of machines m.



(b) Mean objective value for different number of tasks n.

Figure 2.9: Comparison of exact models and the heuristic algorithms.

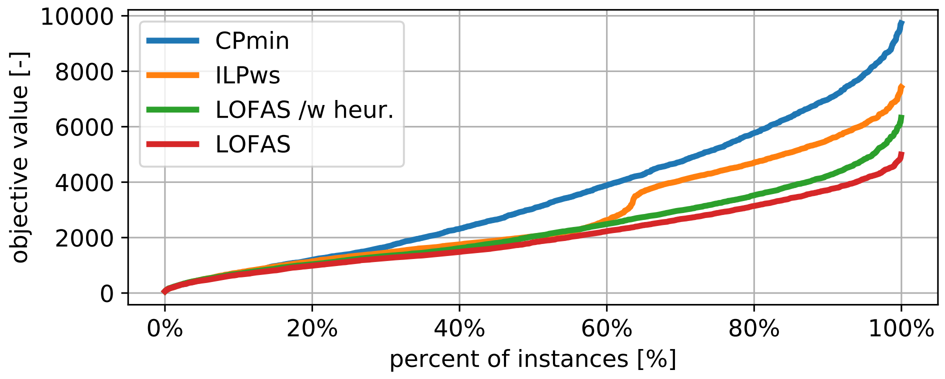


Figure 2.10: Objective distributions of different methods.

to produce any feasible solution to the whole problem, given the sequences on machines. Therefore, LOFAS */w heur.* did not find a solution for any instance with 20 machines, starting from 400 tasks.

To provide further details into the behavior of the algorithm, we report two other statistics. *Dead ends* shows how many times the algorithm hit an infeasible subproblem (due to the constraint on the objective) and restarted to the original solution, and *improv.* reports the number of iterations, where one iteration means avoiding the largest setup on the critical path and solving the subproblem. It can be clearly seen that these two numbers are significantly lower for LOFAS */w heur.* because it almost always wastes all the time allocated to a subproblem, whereas LOFAS is able to save some time on smaller instances, which is efficiently used for exploring more potential improvements. Note that for the instances that were not solved, these numbers are always zero.

Table 2.1: Comparison of exact and heuristic subproblem solvers in LOFAS heuristics.

		LOFAS with exact subproblem			LOFAS with heuristic subproblem		
m	n	objective [–]	dead ends [–]	improv. [–]	objective [–]	dead ends [–]	improv. [–]
5	50	1504.1 (±57.8)	64.6 (±2.4)	134.3 (±51.4)	1517.9 (±55.2)	0.0 (±0.0)	4.9 (±1.8)
10	50	1514.9 (±49.4)	91.4 (±43.4)	349.1 (±145.9)	1545.9 (±32.2)	0.0 (±0.0)	0.8 (±1.2)
15	50	1623.3 (±48.3)	0.0 (±0.0)	15.0 (±31.6)	1871.9 (±24.3)	0.0 (±0.0)	0.0 (±0.0)
20	50	2012.1 (±29.4)	0.0 (±0.0)	3.9 (±6.0)	2485.8 (±45.2)	0.0 (±0.0)	1.6 (±0.7)
5	100	2874.9 (±88.7)	72.2 (±46.2)	247.3 (±127.6)	2939.6 (±83.1)	0.0 (±0.0)	4.8 (±2.5)
10	100	2982.7 (±92.9)	67.9 (±52.0)	189.3 (±108.1)	3063.3 (±70.1)	0.0 (±0.0)	2.3 (±1.8)
15	100	2879.6 (±59.5)	17.9 (±17.1)	77.0 (±28.4)	3279.2 (±34.7)	0.0 (±0.0)	0.0 (±0.0)
20	100	2985.8 (±32.0)	0.0 (±0.0)	15.6 (±19.7)	4035.0 (±50.9)	0.0 (±0.0)	0.0 (±0.0)
5	150	4162.4 (±156.2)	66.5 (±36.1)	125.6 (±62.5)	4258.9 (±150.6)	0.0 (±0.0)	2.4 (±3.1)
10	150	4309.0 (±128.0)	25.5 (±16.8)	79.3 (±40.0)	4415.2 (±120.9)	0.0 (±0.0)	0.7 (±1.3)
15	150	4315.1 (±95.0)	6.4 (±7.2)	43.6 (±13.8)	4673.6 (±72.0)	0.0 (±0.0)	0.0 (±0.0)
20	150	4348.3 (±44.1)	5.6 (±7.3)	22.3 (±7.3)	5456.4 (±60.1)	0.0 (±0.0)	0.0 (±0.0)
5	200	5612.8 (±130.1)	39.6 (±22.8)	67.8 (±26.3)	5707.2 (±126.7)	0.0 (±0.0)	5.1 (±1.9)
10	200	5629.6 (±64.2)	5.6 (±10.2)	30.9 (±18.2)	5734.3 (±56.6)	0.0 (±0.0)	1.4 (±1.5)
15	200	5677.4 (±112.7)	7.6 (±5.4)	25.4 (±6.9)	6032.4 (±64.7)	0.0 (±0.0)	0.0 (±0.0)
20	200	5674.0 (±92.6)	3.4 (±3.6)	10.8 (±3.1)	7023.8 (±61.8)	0.0 (±0.0)	0.0 (±0.0)
5	250	6803.2 (±115.6)	27.6 (±12.4)	40.7 (±9.4)	6903.8 (±115.6)	0.0 (±0.0)	4.8 (±0.4)

10	250	6958.9 (±131.7)	5.8 (±7.2)	19.0 (±5.5)	7100.1 (±125.5)	0.0 (±0.0)	0.6 (±1.3)
15	250	7011.9 (±177.1)	1.3 (±1.5)	7.2 (±3.9)	7417.1 (±45.5)	0.0 (±0.0)	0.2 (±0.4)
20	250	6999.8 (±58.8)	0.6 (±0.8)	2.1 (±1.6)	8339.4 (±107.6)	0.0 (±0.0)	0.0 (±0.0)
5	300	8129.7 (±99.2)	15.2 (±9.1)	20.9 (±11.6)	8246.7 (±91.4)	0.0 (±0.0)	2.3 (±2.5)
10	300	8237.6 (±92.3)	4.7 (±4.1)	8.6 (±4.2)	8384.7 (±54.9)	0.0 (±0.0)	0.6 (±1.0)
15	300	8361.0 (±61.4)	2.4 (±2.4)	5.1 (±2.8)	8707.8 (±61.1)	0.0 (±0.0)	0.0 (±0.0)
20	300	5/10	0.3 (±0.5)	0.4 (±0.5)	9883.1 (±131.9)	0.0 (±0.0)	0.0 (±0.0)
5	350	9719.8 (±177.5)	14.6 (±5.5)	16.3 (±4.3)	9822.0 (±177.5)	0.0 (±0.0)	4.6 (±0.5)
10	350	6/10	4.2 (±4.1)	4.6 (±4.3)	9785.4 (±51.6)	0.0 (±0.0)	1.1 (±1.2)
15	350	2/10	0.1 (±0.3)	0.1 (±0.3)	10417.9 (±96.9)	0.0 (±0.0)	0.0 (±0.0)
20	350	0/10	0.0 (±0.0)	0.0 (±0.0)	2/10	0.0 (±0.0)	0.0 (±0.0)
5	400	8/10	5.0 (±3.4)	6.0 (±4.3)	11234.6 (±96.3)	0.0 (±0.0)	3.4 (±1.3)
10	400	0/10	0.0 (±0.0)	0.0 (±0.0)	11113.3 (±113.8)	0.0 (±0.0)	0.4 (±0.5)
15	400	0/10	0.0 (±0.0)	0.0 (±0.0)	11770.8 (±85.4)	0.0 (±0.0)	0.0 (±0.0)
20	400	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
5	450	8/10	3.2 (±4.7)	3.2 (±4.7)	12276.4 (±157.3)	0.0 (±0.0)	3.2 (±1.7)
10	450	0/10	0.0 (±0.0)	0.0 (±0.0)	12498.1 (±131.5)	0.0 (±0.0)	1.0 (±0.7)
15	450	0/10	0.0 (±0.0)	0.0 (±0.0)	13095.3 (±77.9)	0.0 (±0.0)	0.0 (±0.0)
20	450	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
5	500	3/10	0.8 (±1.3)	1.0 (±1.7)	13910.3 (±226.5)	0.0 (±0.0)	3.3 (±1.3)
10	500	0/10	0.0 (±0.0)	0.0 (±0.0)	13888.2 (±188.1)	0.0 (±0.0)	0.7 (±0.8)
15	500	0/10	0.0 (±0.0)	0.0 (±0.0)	7/10	0.0 (±0.0)	0.0 (±0.0)
20	500	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
5	550	0/10	0.0 (±0.0)	0.0 (±0.0)	15077.0 (±150.9)	0.2 (±0.4)	2.9 (±0.7)
10	550	0/10	0.0 (±0.0)	0.0 (±0.0)	15196.8 (±197.6)	0.0 (±0.0)	0.2 (±0.4)
15	550	0/10	0.0 (±0.0)	0.0 (±0.0)	1/10	0.0 (±0.0)	0.0 (±0.0)
20	550	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
5	600	0/10	0.0 (±0.0)	0.0 (±0.0)	16528.5 (±260.3)	0.2 (±0.4)	3.0 (±1.2)
10	600	0/10	0.0 (±0.0)	0.0 (±0.0)	16706.7 (±327.2)	0.0 (±0.0)	0.2 (±0.4)
15	600	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
20	600	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
5	650	0/10	0.0 (±0.0)	0.0 (±0.0)	17676.3 (±172.8)	0.2 (±0.4)	1.9 (±1.7)
10	650	0/10	0.0 (±0.0)	0.0 (±0.0)	18235.4 (±173.0)	0.0 (±0.0)	0.2 (±0.4)
15	650	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
20	650	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
5	700	0/10	0.0 (±0.0)	0.0 (±0.0)	19114.6 (±317.8)	0.4 (±0.8)	3.2 (±0.4)
10	700	0/10	0.0 (±0.0)	0.0 (±0.0)	19532.0 (±198.8)	0.0 (±0.0)	0.0 (±0.0)
15	700	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
20	700	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
5	750	0/10	0.0 (±0.0)	0.0 (±0.0)	20477.8 (±296.6)	0.1 (±0.3)	2.9 (±0.6)
10	750	0/10	0.0 (±0.0)	0.0 (±0.0)	20774.5 (±123.7)	0.0 (±0.0)	0.2 (±0.4)
15	750	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
20	750	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
5	800	0/10	0.0 (±0.0)	0.0 (±0.0)	21762.7 (±179.1)	1.2 (±1.2)	2.8 (±2.1)
10	800	0/10	0.0 (±0.0)	0.0 (±0.0)	8/10	0.1 (±0.3)	0.5 (±0.7)
15	800	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
20	800	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
5	850	0/10	0.0 (±0.0)	0.0 (±0.0)	23144.0 (±210.5)	1.5 (±1.8)	2.8 (±2.6)
10	850	0/10	0.0 (±0.0)	0.0 (±0.0)	9/10	0.0 (±0.0)	0.0 (±0.0)
15	850	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
20	850	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
5	900	0/10	0.0 (±0.0)	0.0 (±0.0)	24555.4 (±316.8)	1.1 (±1.5)	2.7 (±1.8)

10	900	0/10	0.0 (±0.0)	0.0 (±0.0)	1/10	0.0 (±0.0)	0.0 (±0.0)
15	900	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
20	900	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
5	950	0/10	0.0 (±0.0)	0.0 (±0.0)	25701.5 (±188.7)	0.5 (±0.7)	2.2 (±1.1)
10	950	0/10	0.0 (±0.0)	0.0 (±0.0)	3/10	0.0 (±0.0)	0.0 (±0.0)
15	950	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
20	950	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
5	1000	0/10	0.0 (±0.0)	0.0 (±0.0)	27054.0 (±275.9)	1.7 (±1.6)	3.3 (±1.9)
10	1000	0/10	0.0 (±0.0)	0.0 (±0.0)	1/10	0.0 (±0.0)	0.0 (±0.0)
15	1000	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)
20	1000	0/10	0.0 (±0.0)	0.0 (±0.0)	0/10	0.0 (±0.0)	0.0 (±0.0)

2.5.4 Discussion

We have seen that the performances of the CP models are almost equal with CP3 being the best but its advantage is not significant. Further, the experiments have shown that ILP without a warm start cannot find a feasible solution for instances with $n \geq 8$ tasks reliably, whereas with warm starts it performed better than the best CP model without a warm start. The quality of the solutions from CP with warm starts is much better than ILP with warm starts (even though the warm start for CP is not a complete solution), as can be seen in Figure 2.8. As expected, the heuristic algorithm LOFAS produced the best solutions among all compared methods, although only slightly better than CP3 with warm starts. Smaller instances evidenced that LOFAS achieves objective values quite close to optimal ones. The advantage of LOFAS */w heur.* can be seen in its scalability capabilities as it can solve instances with up to 1000 tasks on 5 machines. On the 50×49 instance set from Section 2.5.1, LOFAS */w heur.* rendered solutions of objective value worse on average by 10.5 % than LOFAS.

2.6 Summary

In this chapter, we tackled the problem of scheduling on dedicated machines with sequence-dependent non-overlapping setups. We suggested an ILP model, five variants of a CP model, and a heuristic approach. The extensive experimental evaluation showed that all exact models themselves are yielding solutions far from optima within the given time limit of 60 seconds, which proved them inappropriate mainly for larger instances. However, the proposed combination of ILP and CP yields high-quality solutions in a very short computation time. The gist is that we leveraged the strength of ILP in the shortest Hamiltonian path problem and the efficiency of CP at sequencing problems with makespan minimization.

The results from this chapter were published in [1] and [2].

3. Optional Activities in Multi-agent Path Finding

There are many practical situations, where a set of agents (robots, cars, etc.) is moving in a shared environment, while each agent is heading for its desired goal position. The environment is usually represented by a graph, where agents can occupy nodes and move along the arcs [43]. The agents are moving cooperatively to avoid collisions and unsolvable congestions. The quality of the sequence of steps leading each agent to the goal position can be measured by some cost function such as makespan.

The problem described above is known as multi-agent path finding (MAPF) [44]. Some examples, where the MAPF problem is useful, include traffic optimization [45, 46], navigation [47], movement in computer games [48], etc.

The state-of-the-art algorithms for the MAPF problem assume that all of the arc lengths are identical and that each node and arc can be occupied by at most one agent at any time. These limitations on the solved problem do not correspond to the reality in many cases. For example, some roads have a larger capacity than others. In this chapter, we add new attributes to the problem specification that bring it closer to the real world.

In particular, we model the MAPF problem in the Constraint Programming (CP) formalism borrowing ideas from scheduling and routing problems. We see the nodes and arcs as resources with limited capacity, which equals one in the typical MAPF setting but can be larger in some applications. We use the modeling concepts introduced in Section 1.4. The motivation is supporting richer (in comparison to traditional MAPF) temporal and capacity constraints.

After formally introducing the classical MAPF problem, we will propose a core scheduling model that allows each agent to visit each node at most once (a single-layer model), as presented in [3]. Then, as in [4], we will extend this model to support multiple visits of each node (a multi-layer model) that complies with the classical MAPF formulation. After that, we will generalize the model to support arcs of different lengths and capacities. While the extension of the proposed scheduling-based model is straightforward for this extended setting, we will also present the extension of the classical SAT-based model. The chapter is concluded by an experimental comparison of scheduling-based and SAT-based models with the goal to find how particular problem attributes influence the efficiency of various models.

3.1 Problem Specification

The MAPF problem is formulated by a graph and a set of agents sitting at certain nodes. The task is to find paths for agents from their origin nodes to their destination nodes while satisfying some constraints, namely, no two agents meet at the same node at the same time, and no two agents swap their positions at one step.

Formally we can define an instance of MAPF as ordered 4-tuple $(G, A, \mathit{orig}, \mathit{dest})$, where $G = (V, E)$ is a directed graph and A is a set of agents.

Functions $orig : A \to V$ and $dest : A \to V$ describe origin and destination nodes of an agent. For each agent $a \in A$, we denote $orig(a) \in V$ the origin location (node) of the agent and $dest(a) \in V$ its destination node.

The solution to a MAPF problem is a sequence of positions in time for each agent that satisfies the conditions that no two agents meet at the same node at the same time, and no two neighboring agents swap their positions at one step. The moves of the agents are discrete and synchronous. In this work, we will focus on solutions that are makespan optimal, i.e., the total time until the last agent reaches its destination is minimized. This requirement of optimality makes the MAPF problem NP-hard [49].

The classical MAPF is usually solved by algorithms that can be divided into two categories:

1. **Reduction-based solvers.** Many solvers reduce MAPF to another known problem such as SAT [50], ILP [51], and answer set programming [52]. These approaches are based on fast solvers that work very well with unit cost parameters.

2. **Search-based solvers.** On the other hand, many recent solvers are search-based. Some are variants of A* over a global search space (i.e., all possibilities how to place agents into the nodes of the graph) [53]. Other make use of novel search trees [54, 55, 56].

All of the above approaches to solving the MAPF problem are designed and tested on graphs with unit-length arcs and unit-capacity arcs and nodes.

3.2 Scheduling Model

This section gives a scheduling-based model for the classical MAPF with unit lengths and unit capacities.

3.2.1 Single-layer Models

We first propose three models restricted such that no agent can visit the same node more than once. Then, we will show how to eliminate this restriction.

Flow Model

The *Flow model* is motivated by the model for the closely related problem of multiple-origin multiple-destination problem [57]. The model consists of two parts, a logical one and a numerical one. The logical part describes a valid path for each agent using the idea of network flows. The numerical part describes temporal and resource constraints, namely that paths for different agents do not overlap in time and space.

For each agent $a \in A$ and for each arc $(x, y) \in E$ we introduce a Boolean decision variable $Used[x, y, a]$ that indicates whether or not arc (x, y) is used to transport agent a. For each agent $a \in A$ and for each vertex $x \in V$ a Boolean variable $Flow[x, a]$ indicates whether or not the transport of agent a goes through the vertex x.

To model a transport path for an agent, we specify the flow preservation constraints. These constraints describe that each agent must leave its origin and must arrive at its destination, and if the agent goes through some vertex then it must enter the vertex and leave it (both exactly once). In the case of origin, the agent only leaves it and, similarly, in the case of destination, the agent only enters it. Formally, for each agent $a \in A$ we introduce the following flow preservation constraints (recall that domains of all the variables are Boolean, that is, $\{0, 1\}$):

$$\forall (x, \mathit{orig}(a)) \in E : \mathit{Used}[x, \mathit{orig}(a), a] = 0 \tag{3.1}$$
$$\forall (\mathit{dest}(a), y) \in E : \mathit{Used}[\mathit{dest}(a), y, a] = 0 \tag{3.2}$$
$$\mathit{Flow}[\mathit{orig}(a), a] = 1 \tag{3.3}$$
$$\mathit{Flow}[\mathit{dest}(a), a] = 1 \tag{3.4}$$
$$\forall x \in V \setminus \{\mathit{orig}(a)\} : \sum_{(y,x)\in E} \mathit{Used}[y, x, a] = \mathit{Flow}[x, a] \tag{3.5}$$
$$\forall x \in V \setminus \{\mathit{dest}(a)\} : \sum_{(x,y)\in E} \mathit{Used}[x, y, a] = \mathit{Flow}[x, a] \tag{3.6}$$

The numerical part specifies non-overlapping constraints, namely two agents do not meet at the same node at the same time, and travel time between the nodes that is expressed by lengths of arcs, which we now consider to be one. To model the time interval when an agent $a \in A$ stays in a node $x \in V$, we introduce two numerical variables $InT[x, a]$ and $OutT[x, a]$ modeling the time when the agent enters the node and when it leaves the node respectively. We can describe the travel time of agent a between the nodes x and y through the arc a as follows:

$$\mathit{Used}[x, y, a] \Rightarrow \mathit{OutT}[x, a] + \mathit{1} = \mathit{InT}[y, a] \tag{3.7}$$

If agent a is going through node x, then the agent cannot enter the node before it leaves it:

$$InT[x, a] \leq OutT[x, a] \tag{3.8}$$

Let $\mathit{sp}(x, y)$ be the length of the shortest path from node x to node y. Then, in order to prune the search space, we can (but do not need to) calculate bounds of the time variables as follows:

$$\forall x \in V \setminus \{\mathit{orig}(a)\} : \mathit{Flow}[x, a] \Rightarrow \mathit{OutT}[\mathit{orig}(a), a] + \mathit{sp}(\mathit{orig}(a), x) \leq \mathit{InT}[x, a] \tag{3.9}$$
$$\forall x \in V \setminus \{\mathit{dest}(a)\} : \mathit{Flow}[x, a] \Rightarrow \mathit{OutT}[x, a] + \mathit{sp}(x, \mathit{dest}(a)) \leq \mathit{InT}[\mathit{dest}(a), a] \tag{3.10}$$

Let *MKSP* be the time when each agent must be in its destination, which corresponds to makespan of the schedule. Then we set the times in the origin and destination of an agent as follows:

$$InT[orig(a), a] = 0 \tag{3.11}$$
$$OutT[dest(a), a] = MKSP \tag{3.12}$$

Finally, to model that two agents p_1 and p_2 do not meet at the same node x, we need to specify that their times of visit do not overlap:

$$(Flow[x, a_1] \wedge Flow[x, a_2]) \Rightarrow$$

$$(OutT[x, a_1] < InT[x, a_2] \lor OutT[x, a_2] < InT[x, a_1]) \tag{3.13}$$

To prohibit swapping the positions of two agents, i.e., to preclude agents from going through the same arc in opposite directions at the same time, we add the following constraint:

$$(Used[x, y, a_1] \land Used[y, x, a_2]) \Rightarrow$$
$$(OutT[x, a_2] < InT[x, a_1] \lor OutT[y, a_1] < InT[y, a_2]) \tag{3.14}$$

Proposition 1. *The Flow model finds a solution to the MAPF problem where no agent visits the same node more than once if and only if such a solution exists.*

Proof. We prove that the Flow model is sound. That is, every consistent instantiation of variables defines a solution to the MAPF problem. The constraints (3.1)-(3.6) define a single path from origin to destination for each agent, i.e., the variables *Flow* and *Used* are equal to one for nodes and arcs used on the path and equal to zero for all other nodes and arcs. The origin and destination must be on the path due to constraints (3.3) and (3.4). The path must continue from origin due to (3.6) and must reach the destination due to (3.5). The path cannot start and cannot finish in any other node due to constraints (3.5) and (3.6). The flow constraints allow a loop to be formed in the graph, but such loops are forbidden by temporal constraints (3.7) and (3.8). Each agent starts its tour at time zero (3.11) and finishes at time *MKSP* (3.12). No two agents can meet at the same node at the same time due to constraint (3.13), and neither can they swap their positions due to constraint (3.14). Hence each solution to the above constraint satisfaction problem defines conflict-free paths for all agents. □

Path Model

The disjunctive non-overlap constraint (3.13) from the *Flow model* is a classical expression of a unary (disjunctive) resource. Recall that in constraint programming, these disjunctive constraints are known to propagate badly and special global constraints modeling resources and efficient filtering algorithms have been proposed [12]. Hence it seems natural to exploit such constraints in a model, where the presence of an agent at a node is modeled as an activity. Activity can be conceived as an interval variable (see Section 1.4). These activities must be connected via temporal constraint to define a path from origin to destination.

Formally, for each agent $a \in A$ and each node $x \in V$, we introduce an activity $N[x, a]$ describing time that the agent a spends in the node x. We denote $\text{StartOf}(N[x, a])$ the start time of the activity, which corresponds to $InT[x, a]$ in the Flow model, and similarly $\text{EndOf}(N[x, a])$ denotes the end time of activity corresponding to $OutT[x, a]$ in the Flow model. The start time of activity corresponding to the origin of the agent is set to zero, while the end time of activity corresponding to the destination of the agent is set to *MKSP*, which is the makespan of the schedule:

$$\text{StartOf}(N[orig(a), a]) = 0 \tag{3.15}$$
$$\text{EndOf}(N[dest(a), a]) = MKSP \tag{3.16}$$

To model the path from origin to destination, we will use a double-link model describing predecessors and successors of activities. The real path will be completed to form a loop by assuming that the origin directly follows the destination. The activities (nodes) that are not used in the path will form self-loops (the node will be its own predecessor and successor).

Formally, for each agent $a \in A$ and for each node $x \in V$ we will use two variables $Prev[x, a]$ and $Next[x, a]$ describing the predecessor and successor of node x on the path of agent a. The domain of the variable $Prev[x, a]$ consists of all nodes y such that $(y, x) \in E$ plus the node x. Similarly, the domain of variable $Next[x, a]$ consists of nodes y such that $(x, y) \in E$ plus the node x. To ensure that the variables are instantiated consistently, we introduce the constraint:

$$Prev[x, a] = y \Leftrightarrow Next[y, a] = x \tag{3.17}$$

To close the loop, we will use the following constraints:

$$Prev[orig(a), a] = dest(a) \tag{3.18}$$

$$Next[dest(a), a] = orig(a) \tag{3.19}$$

It remains to connect information about the path with the activities over the path, namely to properly connect times of the activities so they are ordered correctly in time. This will be realized by the constraint:

$$\text{EndOf}(N[x, a]) + w(x, Next[x, a]) = \text{StartOf}(N[Next[x, a], a]), \tag{3.20}$$

where $w(x, y)$ is the length of arc from x to y, which is 1 for all arcs. However, we set:

$$w(x, x) = -1 \tag{3.21}$$

$$w(dest(a), orig(a)) = -MKSP \tag{3.22}$$

In order to prune the search space, we can (but do not need to) add for all $x \in V \setminus \{orig(a)\}$ the following constraints:

$$Next[x, a] \neq x \Rightarrow \text{EndOf}(N[orig(a), a]) + sp(orig(a), x) \leq \text{StartOf}(N[x, a]), \tag{3.23}$$

and for all $x \in V \setminus \{dest(a)\}$, we add:

$$Next[x, a] \neq x \Rightarrow \text{EndOf}(N[x, a]) + sp(x, dest(a)) \leq \text{StartOf}(N[dest(a), a]) \tag{3.24}$$

For each node $x \in V$, we add the following constraint encoding that the visits of node x are not overlapping[1]:

$$\text{NoOverlap}(\bigcup_{a \in A} N[x, a]) \tag{3.25}$$

Again, to preclude agents from swapping their positions, we need to add the following constraint, for each arc and each pair of agents:

$$\begin{aligned}&(Next[x, a_1] = y \wedge Next[y, a_2] = x) \Rightarrow \\ &\quad (\text{EndOf}(N[x, a_2]) < \text{StartOf}(N[x, a_1]) \vee (\text{EndOf}(N[y, a_1]) < \text{StartOf}(N[y, a_2])\end{aligned} \tag{3.26}$$

[1] A more technical explanation on how the NoOverlap constraint is applied will be given in the Opt model that is described next.

Proposition 2. *The Path model finds a solution to the MAPF problem where no agent visits the same node more than once if and only if such a solution exists.*

Proof. Any solution to the Path constraint model defines a solution of the MAPF problem and vice versa. For each agent, each node (activity) has some predecessor and successor and they are defined consistently thanks to constraint (3.17), i.e., if x is a predecessor of y then y is the successor of x. It means that all nodes of the graph are covered by loops. Moreover, the origin and destination nodes are part of the same loop due to constraints (3.18) and (3.19). All other loops must be of length one due to constraints (3.20) and (3.21). Note that durations of activities are only restricted to be positive numbers and as regular arcs also have positive lengths, the only way to satisfy the constraints (3.20) over the loop is to include an arc with a negative length. Only the arcs (x, x) and $(\mathit{dest}(a), \mathit{orig}(a))$ have negative lengths as specified in constraints (3.21) and (3.22). Finally, each path starts at time zero (3.15) and finishes at time *MKSP* (3.16). No two paths overlap at a node at the same time due to constraint (3.25), and no agents swap their positions due to constraint (3.26). Note that activities that are not used at any path (they are part of loops of length one) are still allocated to unary resource modeling the node. The duration of such activities is one due to constraints (3.20) and (3.21). However, as their start and end times are not restricted by bounds 0 and *MKSP*, such activities can be shifted to future (after *MKSP*). □

Opt Model

The previous model may seem a bit cumbersome in that it is not yet completely devoid of the inefficient disjunctions due to constraint (3.26). This is a clear motivation for the following model that exploits the concept of *optional activities* (see Section 1.4). The succeeding and preceding nodes in a path of an agent will be entailed by whether or not an activity corresponding to the arc and the agent is present. The activities must be connected via temporal constraint to define a path from the origin to the destination.

For each agent $a \in A$ and each node $x \in V$, we introduce three optional activities $N[x, a]$, $N^{out}[x, a]$, and $N^{in}[x, a]$. The activity $N[x, a]$ corresponds to the time of an agent a spent at node x. The activities $N^{in}[x, a]$ and $N^{out}[x, a]$ describe the time spent in the incoming and outgoing arcs. Next, for each agent $a \in A$ and each arc $(x, y) \in E$, we introduce an optional activity $A[x, y, a]$. Notice that all the activities in the model are optional. Finally, we introduce an integer variable *MKSP* to denote the end of schedule (makespan).

The idea is that the path of an agent corresponds to the activities that are present in the solution and that in turn correspond to the nodes and arcs in the path. In the terminology of hierarchical scheduling, it can be conceived such that each activity $N^{out}[x, a]$ has activities $A[x, y, a]$ corresponding to the arcs outgoing from the node x as its children, and symmetrically, $N^{in}[x, a]$ has the activities $A[y, x, a]$ corresponding to the arcs incoming to the node x as its children. Hence, each activity $A[x, y, a]$ has two parents: $N^{out}[x, a]$ and $N^{in}[y, a]$ because the arc (x, y) is an outgoing arc for node x and an incoming arc for node y.

Formally, for each agent $a \in A$, the following logical constraints are introduced:

$$\text{PresenceOf}(N[\mathit{orig}(a), a]) = 1 \tag{3.27}$$

$$\text{PresenceOf}(N[dest(a), a]) = 1 \tag{3.28}$$
$$\text{PresenceOf}(N^{in}[orig(a), a]) = 0 \tag{3.29}$$
$$\text{PresenceOf}(N^{out}[dest(a), a]) = 0 \tag{3.30}$$
$$\forall x \in V \setminus \{orig(a)\} : \text{PresenceOf}(N[x, a]) \Leftrightarrow \text{PresenceOf}(N^{in}[x, a]) \tag{3.31}$$
$$\forall x \in V \setminus \{dest(a)\} : \text{PresenceOf}(N[x, a]) \Leftrightarrow \text{PresenceOf}(N^{out}[x, a]) \tag{3.32}$$
$$\forall x \in V \setminus \{orig(a)\} : \text{Alternative}\left(N^{in}[x, a], \bigcup_{(y,x)\in E} A[y, x, a]\right) \tag{3.33}$$
$$\forall x \in V \setminus \{dest(a)\} : \text{Alternative}\left(N^{out}[x, a], \bigcup_{(x,y)\in E} A[x, y, a]\right) \tag{3.34}$$

The definition of the Alternative constraint, which gets an interval variable as the first argument and a set of interval variables as the second argument, is as follows. If the activity given as the first argument is present, then exactly one activity from the set of activities given as the second argument is present. In addition, it ensures that the start times and end times of the present activities are equal. On the contrary, if the interval variable given as the first argument is absent, then all the interval variables given as the second argument are absent too. Nevertheless, we add the following implication constraints in order to speed-up the search, for each agent $a \in A$:

$$\forall (x, y) \in E : \text{PresenceOf}(A[x, y, a]) \Rightarrow \text{PresenceOf}(N^{in}[y, a]) \tag{3.35}$$
$$\forall (x, y) \in E : \text{PresenceOf}(A[x, y, a]) \Rightarrow \text{PresenceOf}(N^{out}[x, a]) \tag{3.36}$$

The durations of activities N, N^{out}, and N^{in} are to be found, whereas the durations of activities $A[x, y, a]$ are fixed to 1, i.e., $\text{LengthOf}(A[x, y, a]) = 1$. Thanks to the Alternative constraints, the processing times of activities N^{out} and N^{in} will span over the child activity A that will be present, and for the rest, the following constraints need to be added, for each agent $a \in A$:

$$\text{StartOf}(N[orig(a), a]) = 0 \tag{3.37}$$
$$\text{EndOf}(N[dest(a), a]) = MKSP \tag{3.38}$$
$$\forall x \in V \setminus \{orig(a)\} : \text{StartOf}(N[x, a]) = \text{EndOf}(N^{in}[x, a]) \tag{3.39}$$
$$\forall x \in V \setminus \{dest(a)\} : \text{EndOf}(N[x, a]) = \text{StartOf}(N^{out}[x, a]) \tag{3.40}$$

Note that the equality in constraint (3.38) is essential for the correctness of the models. Changing it to an inequality ($\leq$) would mean that the agent disappears after reaching its destination, and thus other agents may use that node, which is prohibited in MAPF.

We need to introduce the constraint precluding the agents from occurring at the same node at the same time, that is, for each node $x \in V$, we add:

$$\text{NoOverlap}\left(\bigcup_{a\in A} N[x, a]\right) \tag{3.41}$$

Recall that the NoOverlap constraint on a set of activities states that it constitutes a chain of non-overlapping activities, any activity in the chain being constrained to end before the start of the next activity in the chain. The NoOverlap constraint uses non-strict inequalities. However, if an agent leaves a node at time

t, another agent is allowed to enter the same node no sooner than at time $t + 1$. In fact, the times spent by agents at nodes are mostly zero (agents go through the nodes without waiting there). Hence, the NoOverlap constraint is given a so-called *transition distance* matrix TDM, which expresses a minimal delay that must elapse between two successive activities. More precisely, $TDM(N_1, N_2)$ gives a minimal allowed time difference between $\text{StartOf}(N_2)$ and $\text{EndOf}(N_1)$. Thus, the constraint (3.41) is given a TDM containing value 1 for each ordered pair of activities from the constraint, which ensures that the time distance between two consecutive visits of a node is at least one. The same construct is in fact used in constraint (3.25).

To prevent agents from using an arc at the same time (swap), we add, for each pair of distinct nodes $x, y \in V$ connected by arc:

$$\text{NoOverlap}\left(\bigcup_{a \in A} \{A[x, y, a], A[y, x, a]\}\right) \tag{3.42}$$

In this case, the transition distance matrix is not needed at all because the default values are 0.

Finally, the objective is to minimize the makespan, i.e., min *MKSP*.

Proposition 3. *The Opt model finds a solution to the MAPF problem where no agent visits the same node more than once if and only if such a solution exists.*

Proof. The solution of the single-layer model consists of selection of activities and their time allocation. The activities corresponding to origins and destinations of agents must be selected due to constraints (3.27) and (3.28). The constraints (3.31)-(3.36) ensure that if a node is used on some path then there must be exactly one incoming and one outgoing arc selected (except for the origin, where no incoming arc is used due to (3.29), and for the destination where no outgoing arc is selected due to (3.30)). No activity outside the path is selected as such activities would have to form a loop due to constraints (3.31)-(3.36), but that would violate the temporal constraints (3.39) and (3.40). Each path starts at time zero (3.37) and finishes at time *MKSP* (3.38). Finally, activities in nodes are not overlapping (3.41), and agents cannot use the same arc at the same time (3.42). □

3.2.2 Multi-layer Model

Now we show how to get rid of the requirement that the agents cannot visit any node more than once. Since in the experimental analysis of the proposed models, which can be found in [3], the Opt model exhibited the most stable performance, we work hereafter only with the Opt model.

In order to let the agents visit the same nodes repeatedly, we simply create a copy of the original graph and add extra arcs to enable transitions between the two graphs. The copies of the original graph will be henceforth referred to as *layers*. Assuming there is a node in which an agent might want to occur ℓ times, we create ℓ such layers. We will show later how ℓ is chosen.

Formally, all the activities in the model are extended with one dimension, corresponding to the layer to which the activity belongs. Now we use $N[x, a, k]$, $N^{in}[x, a, k]$, $N^{out}[x, a, k]$, and $A[x, y, a, k]$, where $k \in \{1, \ldots, \ell\}$ corresponds to the layer. To allow the transitions between two consecutive layers, we introduce for $k \in \{1,\ldots,\ell-1\}$: $A[x,x,a,k]$, which corresponds to transiting an agent a from a node x at layer k to the node x at layer $k+1$. The duration of activity $A[x,x,a,k]$ is set to 0, that is, $\text{LengthOf}(A[x,x,a,k]) = 0$. Note that going through arc $A[x,x,a,k]$ is in fact not a move but merely a transition to another layer. Due to the length 0, an agent can transit an arbitrary number of layers instantly, which is necessary to reach the destination in the final layer ℓ (even if the agent does not re-visit any node).

Also, we need to modify the constraints. The constraints (3.27)–(3.34) are updated, for each agent $a \in A$, to the following constraints:

$$\text{PresenceOf}(N[\mathit{orig}(a),a,1]) = 1 \tag{3.43}$$
$$\text{PresenceOf}(N[\mathit{dest}(a),a,\ell]) = 1 \tag{3.44}$$
$$\text{PresenceOf}(N^{\mathit{in}}[\mathit{orig}(a),a,1]) = 0 \tag{3.45}$$
$$\text{PresenceOf}(N^{\mathit{out}}[\mathit{dest}(a),a,\ell]) = 0 \tag{3.46}$$

$$\forall x \in V, \forall k \in \{1,\ldots,\ell\}, x \neq \mathit{orig}(a) \vee k \neq 1 : \quad \text{PresenceOf}(N[x,a,k]) \Leftrightarrow \text{PresenceOf}(N^{\mathit{in}}[x,a,k]) \tag{3.47}$$
$$\forall x \in V, \forall k \in \{1,\ldots,\ell\}, x \neq \mathit{dest}(a) \vee k \neq \ell : \quad \text{PresenceOf}(N[x,a,k]) \Leftrightarrow \text{PresenceOf}(N^{\mathit{out}}[x,a,k]) \tag{3.48}$$

$$\forall x \in V, \forall k \in \{1,\ldots,\ell\} : \quad \text{Alternative}(N^{\mathit{in}}[x,a,k], \{A[x,x,a,k-1]\} \cup \bigcup_{(y,x)\in E} A[y,x,a,k]) \tag{3.49}$$
$$\forall x \in V, \forall k \in \{1,\ldots,\ell\} : \quad \text{Alternative}(N^{\mathit{out}}[x,a,k], \{A[x,x,a,k]\} \cup \bigcup_{(x,y)\in E} A[x,y,a,k]) \tag{3.50}$$

Note that in order to ease the notational clutter, we neglect the special cases where the transition arc is not defined, i.e., there is no transition arc incoming to the first layer ($A[x,x,a,0]$) and no transition arc outgoing from the layer ℓ ($A[x,x,a,\ell]$). Hence the transition arcs are used (added in the union in constraints (3.49) and (3.50)) only if they are defined.

Constraints (3.35) and (3.36) are modified, and extra implications for transitions are added:

$$\forall (x,y) \in E, \forall k \in \{1,\ldots,\ell\} : \quad \text{PresenceOf}(A[x,y,a,k]) \Rightarrow \text{PresenceOf}(N^{\mathit{in}}[y,a,k]) \tag{3.51}$$
$$\forall (x,y) \in E, \forall k \in \{1,\ldots,\ell\} : \quad \text{PresenceOf}(A[x,y,a,k]) \Rightarrow \text{PresenceOf}(N^{\mathit{out}}[x,a,k]) \tag{3.52}$$
$$\forall x \in V, \forall k \in \{1,\ldots,\ell-1\} : \quad \text{PresenceOf}(A[x,x,a,k]) \Rightarrow \text{PresenceOf}(N^{\mathit{in}}[x,a,k+1]) \tag{3.53}$$

$$\forall x \in V, \forall k \in \{1, \ldots, \ell - 1\}: \quad \text{PresenceOf}(A[x, x, a, k]) \Rightarrow \text{PresenceOf}(N^{out}[x, a, k]) \tag{3.54}$$

Constraints (3.37)–(3.40) are simply changed to the following constraints:

$$\text{StartOf}(N[orig(a), a, 1]) = 0 \tag{3.55}$$

$$\text{EndOf}(N[dest(a), a, \ell]) = MKSP \tag{3.56}$$

$$\forall x \in V, \forall k \in \{1, \ldots, \ell\}, x \neq orig(a) \vee k \neq 1: \quad \text{StartOf}(N[x, a, k]) = \text{EndOf}(N^{in}[x, a, k]) \tag{3.57}$$

$$\forall x \in V, \forall k \in \{1, \ldots, \ell\}, x \neq dest(a) \vee k \neq \ell: \quad \text{EndOf}(N[x, a, k]) = \text{StartOf}(N^{out}[x, a, k]) \tag{3.58}$$

Constraint (3.41) is updated as follows, for each node $x \in V$:

$$\text{NoOverlap}\left(\bigcup_{\substack{a \in A \\ k \in \{1,\ldots,\ell\}}} N[x, a, k]\right) \tag{3.59}$$

And, finally, constraint (3.42) is updated as follows, for each pair of distinct nodes $x, y \in V$ connected by arc:

$$\text{NoOverlap}\left(\bigcup_{\substack{a \in A \\ k \in \{1,\ldots,\ell\}}} \{A[x, y, a, k], A[y, x, a, k]\}\right) \tag{3.60}$$

Recall that the NoOverlap constraint over nodes was given the transition distance matrix TDM ensuring that the time distances between two consecutive visits of a node are at least one. In this case, however, we need to distinguish the time distance of two distinct agents, which must be at least 1, and the time distance of one agent in distinct layers, which must be allowed to be 0 in order for an agent to be able to transit an arbitrary number of layers instantly. More precisely, $TDM(N[x, a, k_1], N[x, a, k_2]) = 0$, and $TDM(N[x, a_1, k_1], N[x, a_2, k_2]) = 1$, for $a_1 \neq a_2$.

Constraint (3.60) does not need any transition distance matrix as the default values 0 are desired.

3.2.3 Algorithm

We presented a constraint model of the problem for a given number of layers. We will show now how to bound the number of layers for a given makespan and then we will present an algorithm for finding the minimal makespan.

Number of Layers

Proposition 4. *Let p_{min} be the shortest path from the origin node to the destination node of an agent, and let MB be an arbitrary upper bound on makespan (can be the optimal makespan). Then, to solve correctly any instance of MAPF problem, it suffices to construct the model with $\frac{MB - p_{min}}{2} + 1$ layers. Moreover, this bound cannot be improved.*

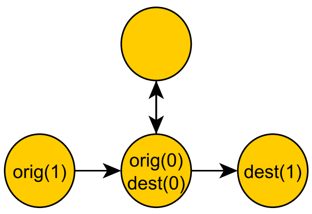


Figure 3.1: Illustration for the proof of Proposition 4.

Proof. The minimum number of nodes that every agent must go through (excluding its origin node) in order to reach its destination node is at least p_{min}. Hence, the maximum number of steps that an agent can spend on repeating already visited nodes is $MB - p_{min}$. However, in order to visit the same node again, an agent must go away from that node and go back to that node, which takes at least two steps. Thus, to get the worst case, an agent has to do cycles of length two to keep visiting the same node. Hence, in the worst case, an agent can revisit the same node as many as $\frac{MB-p_{min}}{2}$ times. Since at least one layer is always required even without repetitions, we obtain the upper bound $\frac{MB-p_{min}}{2} + 1$ on the necessary number of layers.

To show that this bound cannot be improved, we construct a problem such that the necessary number of layers equals the bound (Figure 3.1). Consider an agent 0 having its destination node equal to its origin node ($orig(0) = dest(0)$), hence $p_{min} = 0$, and agent 1 passing that node so that the agent 0 has to jump back and forth (out of his origin node and back). Then the agent 0 occurs in the destination node (including the initial configuration) exactly $\frac{MB-p_{min}}{2} + 1 = \frac{2-0}{2} + 1 = 2$ times.

□

When we know how many layers are needed, we can design the algorithm for solving the MAPF problem. Because all the domains of variables in Constraint Programming (CP) are finite, we need to first obtain an upper bound UB on makespan. Hence, before constructing any model, we first run a polynomial-time algorithm Push and Swap (PaS) [58], which finds arbitrary solution provided the problem is solvable (and assuming $|A| \leq |V| - 2$), and thus we obtain a valid UB. Now, we could calculate the necessary number of layers according to Proposition 4, but because UB obtained from PaS is very loose, the number of layers, as well as the sizes of the domains, would be prohibitively large. That is why we start with one layer and increase the number of layers until we find a solution. This solution might not be makespan-optimal, but it provides better UB which is then used in the final call of the solver that produces a makespan-optimal solution.

The pseudocode of our approach is depicted in Algorithm 3. Calling CP(UB, ℓ) stands for the creation of a model for the problem with ℓ layers and with the upper bound on makespan UB, and solving it with a CP solver. The result of calling CP, as well as calling PaS, is the makespan of the solution, or fail if infeasible. In case of CP, the solution is makespan-optimal with respect to the number of layers.

Proposition 5. *Algorithm 3 finds a makespan-optimal solution to the MAPF problem if and only if a solution exists.*

Algorithm 3 Solving MAPF

```
function SolveMAPF
    $UB \leftarrow PaS$
    if $UB = fail$ then
        return $infeasible$
    end if
    $\ell \leftarrow 1$
    $RET \leftarrow \mathrm{CP}(UB, \ell)$
    while $RET = fail$ do
        $\ell \leftarrow \ell + 1$
        $RET \leftarrow \mathrm{CP}(UB, \ell)$
    end while
    $\ell \leftarrow \frac{RET - p_{min}}{2} + 1$
    return $\mathrm{CP}(RET, \ell)$
end function
```

Proof. Correctness and completeness follow directly from Proposition 3, using the fact that the constraints for the multi-layer model are modified such that the non-conflicting paths from the origin node of each agent, which corresponds to the activity $N[orig(a), a, 1]$, to its destination node, which corresponds to the activity $N[dest(a), a, \ell]$, are found whenever a solution exists for the number of layers ℓ. Since the CP solver always returns the makespan-optimal solution for a given number of layers and using the fact that the problem is ultimately modeled with the sufficient number of layers according to Proposition 4, the obtained solution is optimal. □

Note that we also tried different variants of the multi-layered model. For example, we tested the models where the transitions of agents between layers are synchronized, and hence the NoOverlap constraints are imposed only within one layer instead of over all layers. However, these models turned out to be less efficient in that they require a higher number of layers to maintain optimality.

3.3 Generalized MAPF

Recall that the main motivation for using the scheduling-based model was its applicability to more general problems. One such generalization of MAPF is labeling the arcs with lengths (weights), which determine the duration of going from one node to another, and with capacities determining the number of agents that can occur at one arc at the same time (referred to as *occupancy*). Now we work with an arc-weighted graph $G = (V, E, w, occ)$, where $w(x, y)$ indicates the duration of moving an agent over the arc (x, y), and $occ(\{x, y\})$ stands for how many agents can use the pair of arcs (x, y) and (y, x) at the same time.

Using non-directional occupancy allows us to model the original MAPF problem with prohibited swaps of agents (by setting occupancy to 1), which would not be possible with directional capacities. The motivation is that a road capacity is shared in both directions, while the travel time can be different (e.g., uphill/downhill).

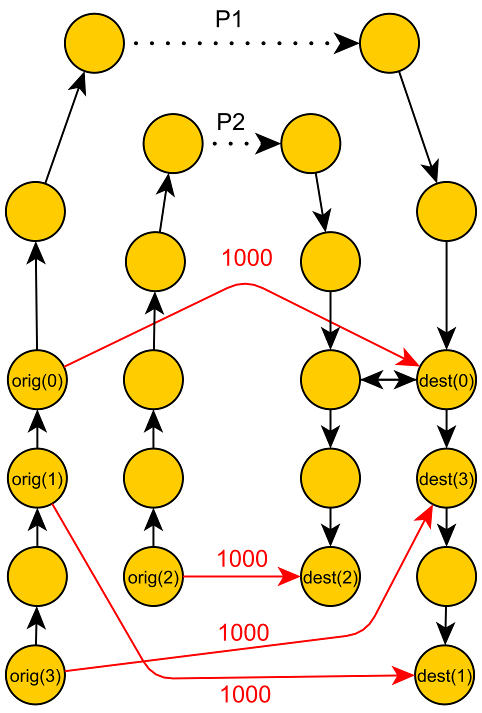


Figure 3.2: Pathological example for SM-OPT.

3.3.1 Scheduling-Based Approach

We model the generalization of MAPF using the multi-layer model described in the previous section, where the NoOverlap constraints over arcs (3.60) are substituted with cumulative functions [26] with capacities set to corresponding $occ(\{x, y\})$, and the duration of activities corresponding to arcs are set to the weights of arcs, i.e., $\text{LengthOf}(A[x, y, a, k]) = w(x, y)$.

Let us refer to the Algorithm 3 using the model with these generalizations as SM-OPT. It is easy to verify that the propositions from the previous section can be used also for the generalized MAPF. Clearly, increasing *occ* can only improve the makespan. However, the bound on makespan MB and the shortest path of an agent p_{min} in Proposition 4 must be measured with respect to the lengths of arcs. Measuring MB and p_{min} w.r.t. the number of arcs would be incorrect. To see this, consider the situation in Figure 3.2.

Each origin node of any agent is directly connected to its destination node with one arc of a very large length, say 1000, which is depicted in red. The other black solid arcs are unit-length, while the dotted arrows P1 and P2 stand for very long paths consisting of unit-length arcs (and the corresponding number of nodes). Suppose the length of the shortest path from $orig(i)$ to $dest(i)$, for $i \neq 0$, is 500 (by the paths P1 and P2), and the length of the shortest path from $orig(0)$ to $dest(0)$ is 496. Hence, $p_{min} = 496$, and the minimal makespan is 500, which is achieved via paths P1 and P2 with the caveat that agent 0 reaches its destination first and then he has to jump back and forth to the neighboring node so as to clear the way for the other agents.

What we get from the PaS algorithm in this example is the solution where each agent goes over the direct arc to the destination, hence the obtained makespan is 1000. If we treated the result w.r.t. the number of arcs, that is, $MB = 1$ and

$p_{min} = 1$, we would get $\frac{1-1}{2} + 1 = 1$, so that the algorithm would settle for the solution with just one layer, which is far from optimum (we can set arbitrarily long arcs instead of 1000).

Now, if we use the Proposition 4 correctly with the makespan bound MB and shortest path p_{min} measured w.r.t. the length of arcs, we could compute the number of layers as $\frac{500-496}{2} + 1 = 3$, which is exactly the number that is necessary because agent 0 occurs in its destination exactly three times. This example also confirms that the bound on the number of layers is tight.

Nevertheless, since we obtain the solution of makespan 1000 from the PaS algorithm as well as from the first run of CP(1000, 1), the SM-OPT computes the final number of layers as $\ell = \frac{1000-496}{2} + 1 = 253$, which is unnecessarily too much. The question how to improve this bound remains open.

3.3.2 Heuristic Approach

The main problem with the generalized MAPF is that the number of layers calculated using the Proposition 4 may be very high. In order to observe the potential of the scheduling-based approach, let us define SM-HEUR as Algorithm 3 modified such that it starts from one layer and increases the number of layers by one only until a feasible solution is found and this solution is returned to the user. Formally, the last call to the solver at line 13 is not realized and instead, the algorithm returns RET calculated in the loop. Clearly, this does not guarantee to find a makespan-optimal solution, and as the pathological case from Figure 3.2 shows, it can be arbitrarily far from the optimum.

3.3.3 SAT-Based Approach

While the generalizations described above are easy to implement in our scheduling-based model, the same generalizations can be very challenging (both in implementation and runtime) in other existing approaches. One of the most popular approaches is reducing the MAPF problem to a SAT formula [50]. Jiří Švancara implemented such solver using the Picat language, which has been showed to be comparable with the state-of-the-art SAT-based MAPF solver [59], and also implemented a version for the generalized MAPF. A detailed description of this approach can be found in [4].

3.4 Experiments

We implemented the scheduling approach in the IBM CP Optimizer version 12.8 [11]. The only parameter that we adjusted is `Workers`, which is the number of threads the solver can use and which we set to 1. For the SAT-based approach, we used the Picat language and compiler version 2.2#3 [59]. The experiments were run on a PC with an Intel® Xeon™ CPU E5-2660 v2 running at 2.00 GHz with 16 GB of RAM. We used a cutoff time of 1000 seconds per problem instance.

3.4.1 Implementation Details

We first compute the all-pairs-shortest-path matrix *sp* using the *Floyd-Warshall* algorithm [60] as the preprocessing phase. We set the lower bound on makespan to be the maximum, over all agents, of the shortest paths from the origin node to the destination node of the agent.

To represent the activities in the model, we use the *interval variables* of the CP Optimizer, which are designed for the scheduling problems and support specialized constraints such as Alternative and NoOverlap. The bounds of the intervals and other time variables are limited using the *sp* matrix, namely, $\forall a \in A$, $\forall k \in \{1, \dots, \ell\}$, $\forall x \in V$, the lower bound on start time (EST) of $N[x, a, k]$ is set to $sp(orig(a), x)$, and the upper bound on end time (LCT) of $N[x, a, k]$ is set to $UB - sp(x, dest(a))$. Further, if $EST > LCT$, it means that the node x cannot be passed through by agent a, and thus we omit creating variables associated with node x and agent a.

3.4.2 Problem Instances

The problem instances are created over strongly biconnected undirected graphs. These types of graphs ensure that the instance is always solvable as long as there are at least 2 agents less than the number of nodes [61]. To create the different complexity of the instances, we incrementally increase the number of nodes in the graph (from 20 nodes to 40 nodes with the increment of 5) as well as the number of agents in the graph (from 2 to 9 agents). Both the origin and destination positions of agents are randomly placed in the graph.

Further, we added lengths to the arcs. The length of each arc is chosen uniformly at random from the range $[1, W]$, where $W \in \{1, 50, 100, 200, 300\}$. Occupancy is a global attribute for the instance and is also incrementally increased (from 1 to the number of agents in the instance). Altogether we generated 1100 instances.

3.4.3 Results

The charts show the number of problems solved (x-axis) within a given time (y-axis). Hence, a curve that is closer to the bottom right represents a better method. All of the charts have a logarithmic scale on the y-axis.

We compare three methods described above: SM-OPT, SM-HEUR, and the SAT-based model (labeled as ”Picat”). Recall that SM-HEUR does not guarantee to find an optimal solution.

The overall comparison of SM-OPT, SM-HEUR, and Picat is depicted in Figure 3.3. In Figures 3.4–3.8, we show comparisons for the given maximum length of arcs W to see how the lengths of arcs affect the efficiency of the approaches.

It can be noted that when all arcs are unit-length, which is closest to the original MAPF problem, Picat is positively faster than the scheduling-based methods (Figure 3.4). However, with increasing upper bound on arc length (and thus increasing the differences in individual arc lengths in an instance), the scheduling-based methods are becoming faster. This can be seen with the maximum length of 50 (Figure 3.5), where SM-OPT is comparable with Picat, and with the maximum length of 100 (Figure 3.6), where SM-OPT is faster than Picat.

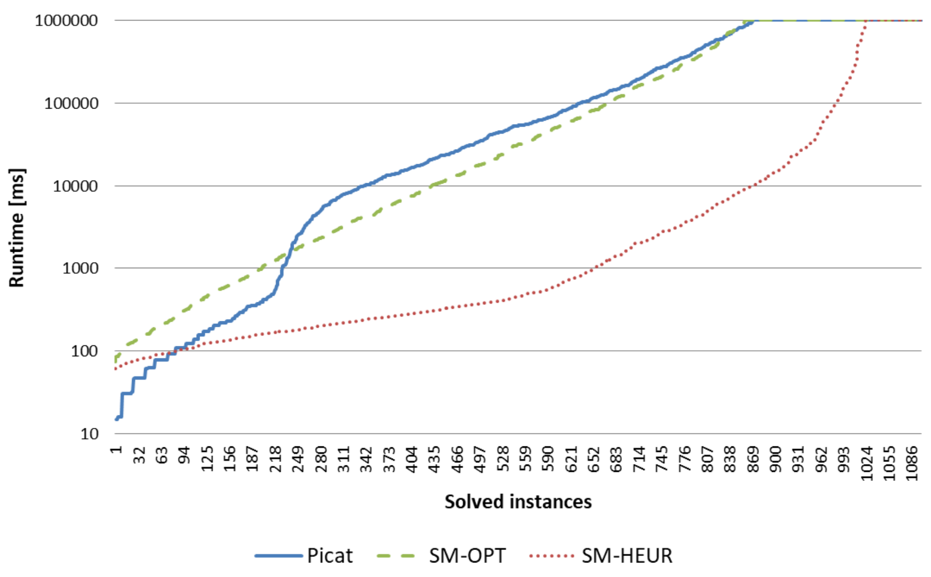


Figure 3.3: Comparison of all generated instances.

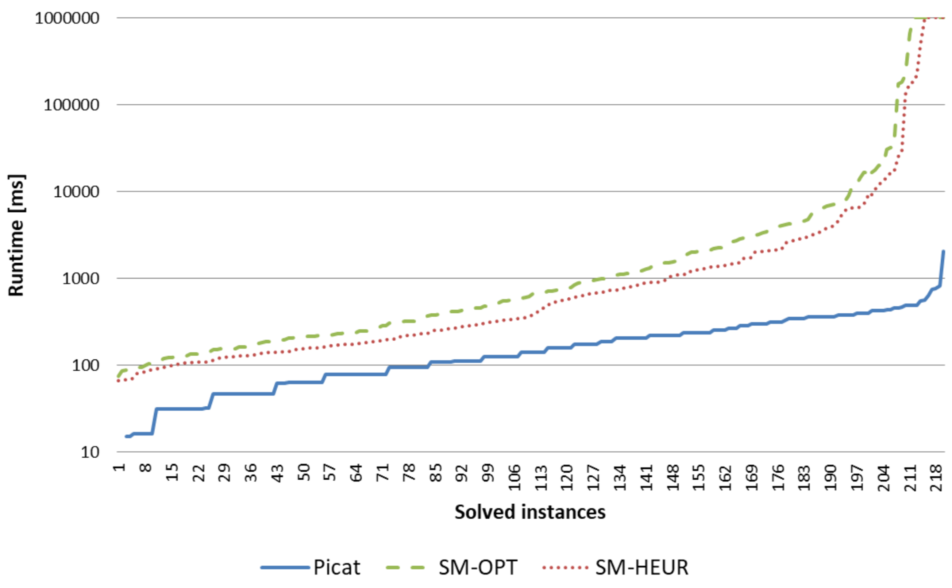


Figure 3.4: Instances with the maximum length of 1.

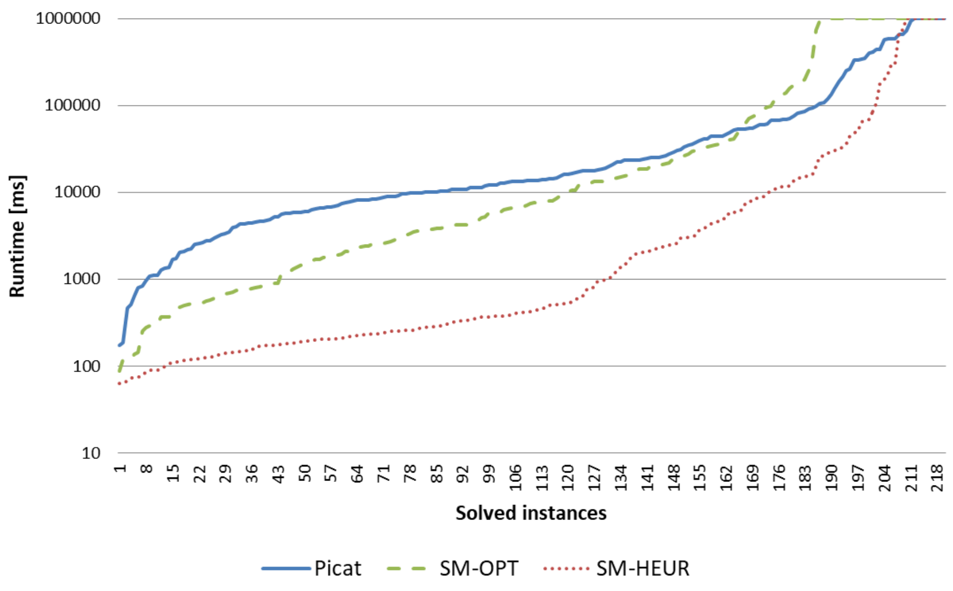


Figure 3.5: Instances with the maximum length of 50.

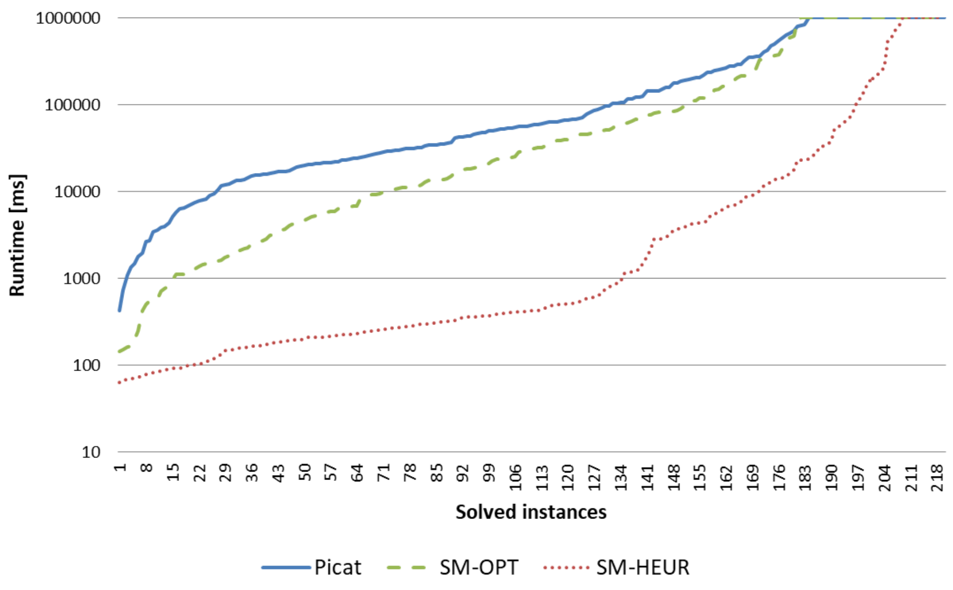


Figure 3.6: Instances with the maximum length of 100.

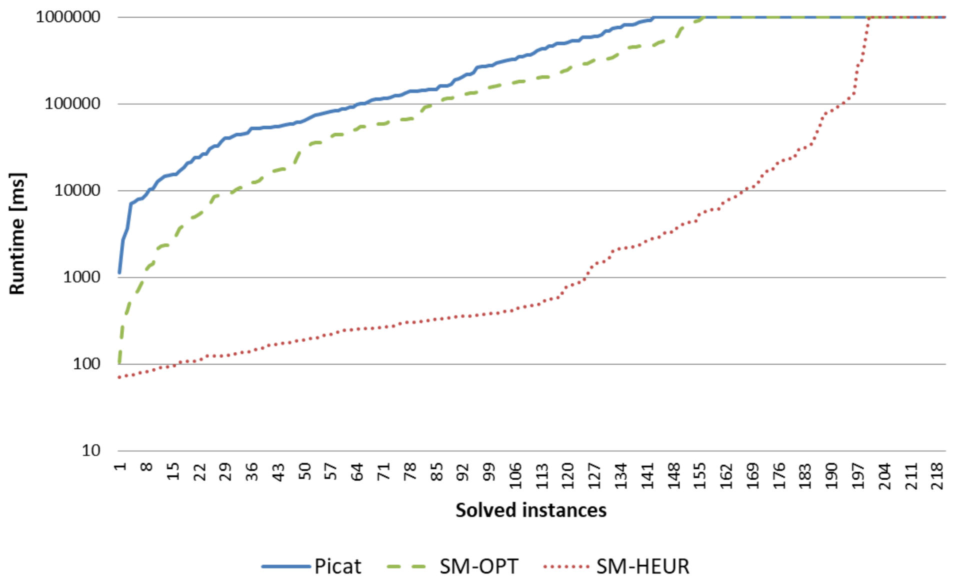


Figure 3.7: Instances with the maximum length of 200.

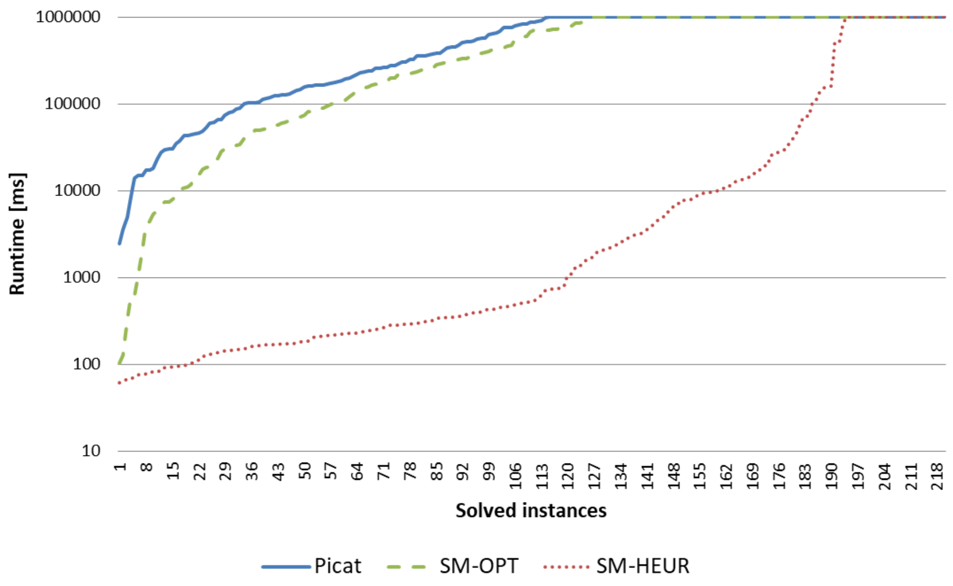


Figure 3.8: Instances with the maximum length of 300.

max length	1	50	100	200	300
1 *s*	0.59	5.38	7.50	inf	inf
10 *s*	0.89	1.47	2.73	3.88	4.00
100 *s*	0.94	0.94	1.15	1.29	1.73
1000 *s*	0.96	0.89	0.99	1.09	1.11

Table 3.1: Ratio of solved instances of SM-OPT to Picat within the given time limit.

In addition, Table 3.1 shows the ratio of the number of solved instances by SM-OPT to the number of solved instances by Picat, within the selected time limit (1–1000 seconds), for the given upper bound on arc length. In the first row, inf means that Picat did not solve any instance in one second.

The results clearly confirm the hypothesis that with the increasing length of arcs, the advantage of SM-OPT over Picat is increasing, which is even more apparent for lower time limits.

Figure 3.9 shows the comparison on instances with the maximum length of 1 and arc occupancy only of 1, which is the classical MAPF problem. The similarity of the charts in Figure 3.9 and in Figure 3.4 shows that the occupancy parameter does not have a significant impact on the efficiency of any approach.

The results also show that the SM-HEUR is by orders of magnitude faster than SM-OPT, while the number of problems where it found a sub-optimal solution is 71 out of 860. However, the average (over these 71 sub-optimal answers) increase in the makespan is by 25.34 % from the optimum. This should be an incentive for further research on the number of re-visits of an agent at a node.

Another advantage of our technique over the SAT-based approach is that even if it does not finish in time, it can find at least some solution. Out of the 240 instances that SM-OPT did not solve in the given time limit, it found a feasible solution in 195 cases, i.e., it did not find any solution only for 45 instances. This contrasts with Picat which either finds an optimal solution or nothing. Thus, Picat did not find any solution for 230 instances.

3.5 Summary

We showed how to model a MAPF problem as a scheduling problem, where nodes and arcs are seen as resources used by the agents. First, we modeled the classical problem often used in the literature (unit lengths and capacities). Then we extended this problem by introducing arc length and arc occupancy limits to simulate real-world conditions. We showed that this extension is easy to model in our scheduling approach, but it is more challenging for a classical SAT-based approach.

We compared the discussed approaches experimentally. As expected, the classical SAT approach outperforms the scheduling-based methods on the classical MAPF problem instances. However, with the increasing lengths of arcs, our scheduling-based techniques outperform the SAT-based approach.

The hardness of the problem can be linked to the number of layers needed for its solution. This is true for both the scheduling approach and the SAT approach. One timestep is equivalent to one layer in the SAT approach and therefore we

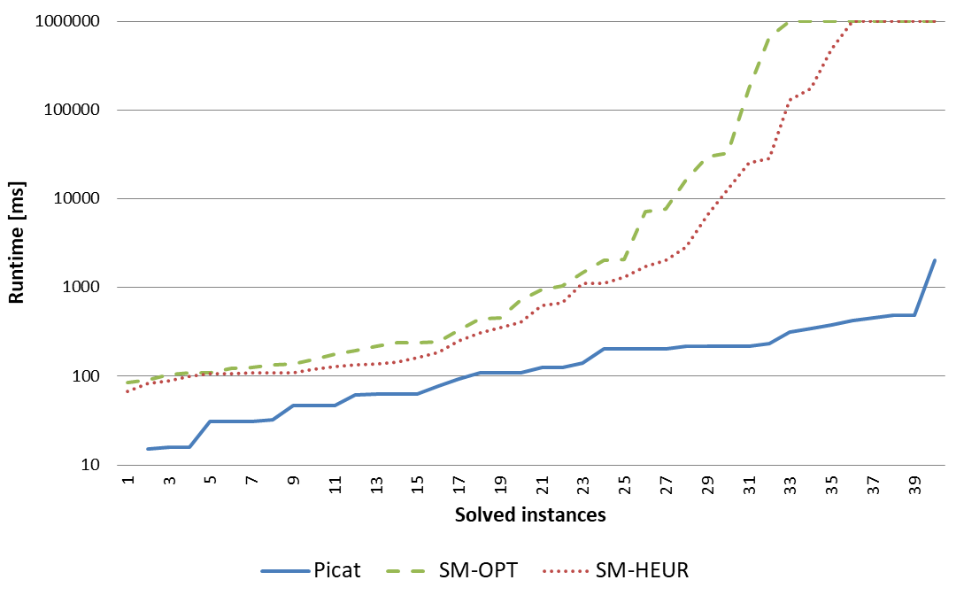


Figure 3.9: Instances with the maximum length of 1 and maximum arc occupancy of 1.

cannot improve on the number of layers needed. On the other hand, the number of layers in the scheduling approach is equivalent to the number of returns to a single node. We created a formula that estimates the number of layers needed. In many cases, however, this number is greatly overestimated.

The single-layer models were proposed and evaluated in [3], and the multi-layer extension was presented in [4]. Further, we also investigated the dynamic version of MAPF, where agents appear online [5].

4. Optional Activities in Time-Sensitive Networking

Real-time communication in automotive and industrial domains requires guaranteeing bounded jitter and end-to-end latency of the data being sent. The deterministic behavior of the communication network is achieved by time-triggered (TT) schedules calculated off-line for critical data traffic. These schedules are required to satisfy several constraints, e.g., flow precedence constraints or resource constraints. Omitting some constraints would yield communication schedules that would be impossible to realize (execute). Further, as elaborated in [62] or [8], it is also necessary to consider the frame isolation constraint, which basically means that there can never be more than one frame of TT traffic stored in a queue at any time. Otherwise the deterministic behavior would not be guaranteed since a loss of one frame could cause delays in the delivery times of other frames.

In practice, it is often desired to increase the number of communication flows while maintaining the network size. Hence, crafting a schedule for all the communication flows such that all the constraints are satisfied is often impossible. In the current approaches to scheduling TT communication, the routes of flows are almost always fixed or computed before the scheduling process. If this assumption is relaxed and the routes are to be determined jointly with scheduling, more flows could likely be scheduled as shown in [63]. In this chapter, motivated to increase the schedulability and throughput of the networks, we address the problem of joint routing and scheduling (JRaS) in IEEE 802.1Qbv Time-Sensitive Networks (TSN). We focus on the isochronous type of traffic [64], where zero-jitter and deadline requirements must be met.

Suppose an optimal solution to scheduling TT traffic in the Ethernet-based networks (including TSN) is desired. In that case, the problems are usually modeled in Satisfiability Modulo Theories (SMT) formalism or sometimes as Integer Linear Programs (ILP). In this chapter, we propose two models based on Constraint Programming (CP). All these three modeling paradigms are described in Chapter 1. Next, we deduce approaches based on Logic-Based Benders Decomposition and pruning the search space by restricting the scheduling problem. Finally, we give an extensive experimental evaluation of the proposed methods and the existing methods from the literature that we further improved.

The main contributions of this chapter, which were presented in [6], are:

- two CP models: one with simple disjunctions and one with optional interval variables that represent waiting of frames in queues,
- application of the Logic-Based Benders Decomposition on the proposed models, as well as on the state-of-the-art models,
- benchmarking and experimental evaluation of the suggested solution techniques, showing the superiority of the proposed CP models, especially in combination with the Logic-Based Benders Decomposition, increasing the schedulability by more than 50 % over the state-of-the-art methods.

4.1 Related Work

The need to solve routing and scheduling jointly emerges in various domains, e.g., in healthcare [65], vehicle routing with cross-docking [66], ship routing of oil companies [67], school bus routing [68], and so on. However, the proposed methods cannot be applied to the problem at hand due to its periodicity and other specific constraints.

The work that considers JRaS of TT traffic in TSN networks is scarce. Caddell [69] proposed an SMT model for JRaS and gave an experimental evaluation of the implementations of the model in SMT (using Z3) and ILP (using CPLEX). However, this model does not consider the issue of isolation of frames.

JRaS in automotive TT networks is addressed in [70]. The authors model the problem as a set of pseudo-boolean constraints, which are then solved by pseudo-boolean solver SAT4J, but the frame isolation is neglected.

Falk et al. [71] propose an ILP model for JRaS, enforcing the no-wait constraint, i.e., the frames are not allowed to wait at switches in the queues at all, which is more restrictive than necessary. The model considers the decision problem without any objective, and the experiments evaluate the computation time of the solver.

The routing algorithms for TSN TT communication and their impact on schedulability are investigated by Nayak et al. [72]. In this case, again, the scheduling algorithms treat the problem as no-wait packet scheduling where the frames are not allowed to wait in the queues at all [73].

Atallah et al. [74] focus on calculating multicast routes and schedules also enforcing the no-wait constraint. The authors propose a time-indexed formulation in ILP. This model is incorporated in the iterated scheduling approach, where the flows for consecutive scheduling iterations are grouped such that the degree of conflict among the groups is minimized.

Another ILP model for multicast routing and scheduling is presented by Yu and Gu [75]. This model is somewhat similar to that of [69], but the optimization objective is to balance the traffic load in the network. The authors propose link grouping of the complete subgraphs in the network as a pre-processing phase and splitting it back to the original topology when not schedulable as a post-processing phase. However, the ILP model is invoked iteratively, always adding just one new flow without having the option to modify already scheduled flows.

JRaS of TT traffic on TTEthernet is addressed by Schweissguth et al. [63]. The authors showed that considering routing and scheduling jointly in one model may schedule problem instances with high utilization that are otherwise unschedulable when routing is fixed first, but then showed that it is counterproductive in some cases due to the significant increase of the search space. The same authors [76] updated their approach to address JRaS of TT traffic for TSN, where the model is extended to allow for multicast routing. However, the paper seems to disregard some concepts of TSN, such as the usage of FIFO queues.

TT traffic on TTEthernet along with Audio Video Bridging (AVB) traffic is considered by Pop et al. [77]. First, the authors take the same frame scheduling model as in [62] and implement it in ILP solver CPLEX. Once the schedule for TT messages is fixed, the authors propose a heuristic algorithm for optimizing the routes of AVB flows, where the objective concerns minimization of worst-case delay. The heuristic, which was initially presented in [78] without considering TT traffic, is based on Greedy Randomized Adaptive Search Procedure (GRASP) [79].

The combination of AVB and TT traffic in TSN is studied by Gavrilut et al. [80]. The authors use a classical decomposed approach to JRaS where routing and scheduling TT flows are solved separately, both by heuristic approaches. While scheduling is done by GRASP, routing is done using the K-Shortest Paths algorithm [81] to acquire a reduced set of potential paths, from which the least utilized path is then selected. A greedy heuristic approach taking the least utilized paths to balance the traffic load is also investigated by Ojewale and Yomsi [82].

The issue of routing has also been studied by Nayak et al. [83, 84]. The authors address a time-sensitive software-defined network (TSSDN), a network architecture based on software-defined networking (SDN), compliant with IEEE 802.3 standard. Data transmissions in TSSDN are scheduled only on the end-stations and not on the switches in the whole network. The authors propose several variants of ILP formulations that solve the combined problem of routing and scheduling [83] and online scheduling ILP-based algorithms for computing incremental schedules at runtime [84].

4.2 Problem Specification

Switches compliant with the IEEE 802.1Qbv standard (Figure 4.1) contain first-in-first-out (FIFO) queues where the frames are saved before they are forwarded. Each such queue is linked to a gate that is open or closed. When the gate is closed, the sending of frames from the linked queue is disabled. When the gate is open, the frames are sent to the egress port in the same order as they have been enqueued (that is, FIFO). If more gates are open simultaneously, the so-called time-aware shaper (TAS) transfers the frames from the queue of the highest priority. The static periodic schedule stipulating the intervals when the gates are closed or open is called *gate control list* (GCL).

Opening exclusively one queue at a time results in a time-deterministic performance of the network. Instead, we calculate a GCL such that at any time, either one queue for the TT traffic is open and all the other queues are closed, or all the queues are open, but the queues for the TT traffic are closed. The classes of non-scheduled traffic are then prioritized by TAS. In this method, there is no intrusion of the non-scheduled traffic into the TT traffic as all the other queues are blocked whenever it comes to sending a TT frame. Thus, the deterministic execution and satisfaction of application constraints (such as latencies and jitter) for the TT traffic are guaranteed.

The aim is to obtain a GCL, a list of time intervals during which the timed gate is open, for each queue on each egress port. We focus on the scheduling of the TT traffic, and we adhere to the strategy as in [62], which is to find the transmission offsets for each frame as well as the assignment to which queue the frame is saved. The GCLs are then calculated in the postprocessing phase.

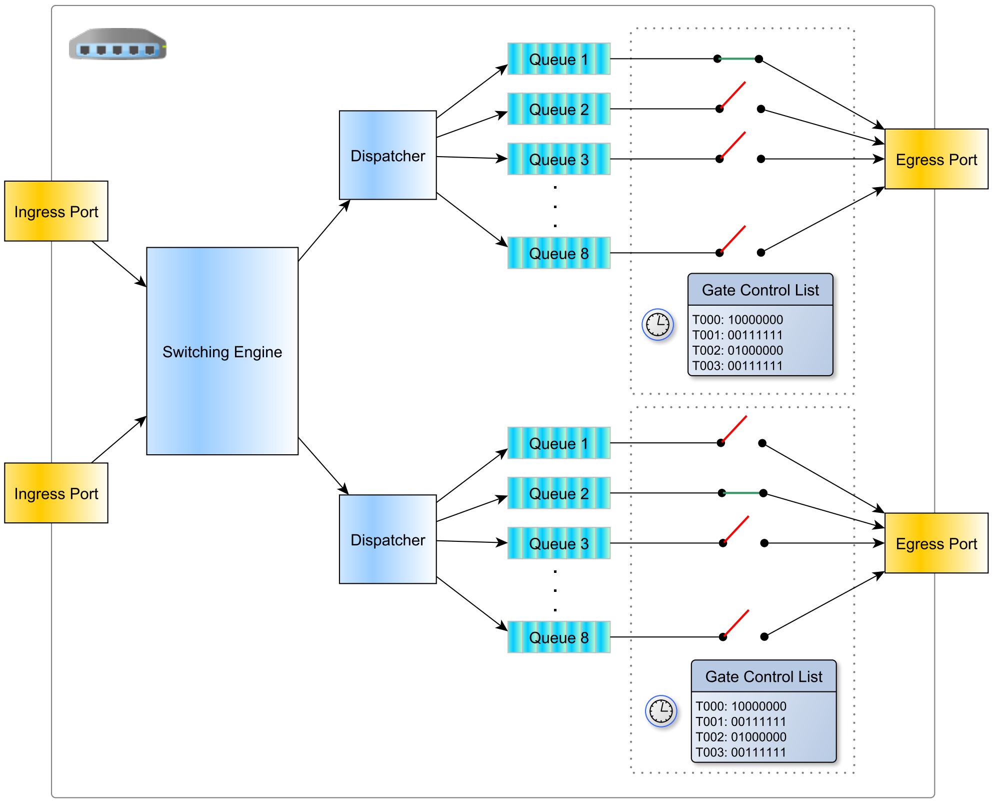


Figure 4.1: Scheme of an IEEE 802.1Qbv TSN switch.

4.2.1 Network Model

We model the network as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where set of nodes $\mathcal{V}$ embodies the switches and the end-stations (such as sensor, actuator, camera, or programmable controller), and set of edges $\mathcal{E}$ embodies the physical connections (Ethernet links) between the nodes. Link $(a, b) \in \mathcal{E}$ expresses one direction of the communication, hence, a pair of links $(a, b), (b, a) \in \mathcal{E}$ embodies the full-duplex physical connection between nodes a and b. From the scheduling point of view, these two links comprise two distinct resources.

Each link $(a, b) \in \mathcal{E}$ is characterized by its speed $c_{a,b}$, the number of queues available for TT traffic $q_{a,b}$, and link delay $d_{a,b}$, which is the propagation delay on the medium and can be calculated as the wire length divided by the speed of light. Each switch $a \in \mathcal{V}$ is also specified by its switching delay d_a.

We assume all the devices in a network are time-synchronized. The worst-case synchronization error, i.e., the maximum difference between the local clocks of any two devices in the network, is denoted as δ.

4.2.2 Application Model

Communication in the network is realized using the notion of flows. A flow is a periodic data transmission from one end-station to another end-station. The set of flows is denoted as $\mathcal{F}$. Each flow $i \in \mathcal{F}$ is specified by talker t_i, which is the end-station ID that is sending the data, listener l_i, which is the end-station ID that receives the data, payload p_i in bytes, period (data cycle) T_i, relative deadline $\tilde{d}_i$, which is the maximum time till when the listener must receive the data, and relative release date r_i, which is the minimum time when the talker can start sending the data. It must hold that $0 \leq r_i \leq \tilde{d}_i \leq T_i$. The reason why the deadline of a flow cannot exceed the period of the flow is linked to the functionality of a controller driving a physical plant. The cycle of such a controller consists primarily of two parts: calculation and receiving input or sending output data. Both these parts must be finished in the period of the corresponding flow.

Each flow is assumed to transmit no more data than what can fit into one Ethernet frame of the Maximum Transmission Unit (MTU) size (1542 bytes). The flows may have different periods, which may end up with the hyperperiod (network cycle) larger than the period of any flow. Let us denote hyperperiod HP as the least common multiple (lcm) of the periods of all flows: $HP = \text{lcm}\{T_i \mid i \in \mathcal{F}\}$. Hence, one flow may occur more than once in the hyperperiod. These occurrences of a flow on a link are referred to as frame occurrences.

We assume that zero jitter (strictly periodic schedule) is required, i.e., the time distance of every subsequent frame occurrence of flow $i \in \mathcal{F}$ is always equal to T_i. Zero jitter is a standard requirement in systems with feedback control where input or output signals are propagated with equidistant signal sampling.

Table 4.1: Notation

symbol	**meaning**
$\mathcal{V}$	set of switches and end-stations
$\mathcal{E}$	set of links
$c_{a,b}$	speed of link (a,b)
$q_{a,b}$	number of queues available on the egress port of link (a,b)
$d_{a,b}$	delay of link (a,b)
d_a	switching delay of switch a
δ	maximum synchronization error
$\mathcal{F}$	set of flows
t_i	talker of flow i
l_i	listener of flow i
p_i	payload of flow i
T_i	period of flow i
$\tilde{d}_i$	deadline of flow i
r_i	release date of flow i
$L_{i,a,b}$	transmission duration of a frame of flow i on link (a,b)
HP	hyperperiod
$\rho_{i,a,b}$	true if flow i is routed through link (a,b) (variable)
$\phi_{i,a,b}$	transmission offset of the first frame occurrence of flow i on link (a,b) (variable)
$\lambda_{i,a,b}$	queue ID to which the frame occurrences of flow i on link (a,b) are stored (variable)

We introduce, $\forall i \in \mathcal{F}, \forall (a,b) \in \mathcal{E}$:

1. boolean variable $\rho_{i,a,b}$ indicating whether or not flow i is routed through link (a,b),

2. integer variable $\phi_{i,a,b}$ representing the transmission offset of the first frame occurrence of flow i on link (a,b), and

3. integer variable $\lambda_{i,a,b}$ representing the queue ID to which all the frame occurrences of flow i on link (a,b) are stored.

Note that variables $\phi_{i,a,b}$ and $\lambda_{i,a,b}$ are relevant only if variable $\rho_{i,a,b}$ is true. Finally, let $L_{i,a,b}$ be the time it takes to transmit any frame occurrence of flow $i \in \mathcal{F}$ over link (a,b), which is referred to as transmission duration and is calculated as $L_{i,a,b} = \frac{8 \cdot p_i}{c_{a,b}}$. The notation is summarized in Table 4.1.

4.2.3 Example

To elucidate the problem at hand, consider the following problem instance. As illustrated in Figure 4.2, the network consists of nodes $\mathcal{V} = \{1, 2, 3, 4, 5, 6, 7, 8\}$ and links $\mathcal{E} = \{(1, 6), (6, 1), (2, 6), (6, 2), (3, 7), (7, 3), (4, 8), (8, 4), (5, 8),$ $(8, 5), (6, 7), (7, 6), (6, 8), (8, 6), (7, 8), (8, 7)\}$.

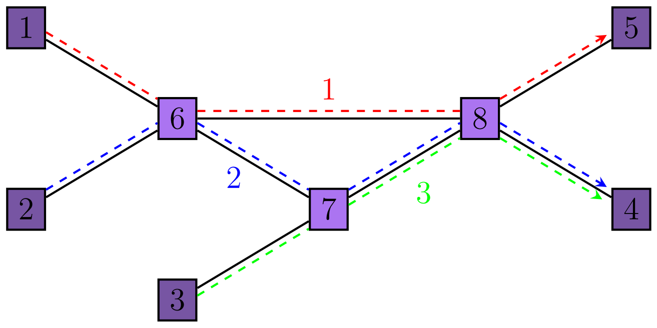


Figure 4.2: Illustration of the network and the flows. Network nodes are depicted by rounded squares, links by solid black edges between the nodes, and flows by dashed lines.

For each link $(a, b) \in \mathcal{E}$, the number of available queues for TT traffic is $q_{a,b} = 1$ and the propagation delay is $d_{a,b} = 0.17\mu s$, and for each node $a \in \mathcal{V}$, the switching delay is $d_a = 0.5\mu s$. The maximum synchronization error is $\delta = 0.1\mu s$. All the time units are in microseconds so we henceforth omit writing "μs".

The parameters of the flows from the set of flows $\mathcal{F} = \{1, 2, 3\}$ are listed in Table 4.2. An example of a routing and a schedule to be found by the solver is represented by a Gantt chart in Figure 4.3. The periods of flows result in hyperperiod $HP = \text{lcm}\{150, 100, 100\} = 300$. Hence, flow 1 occurs twice in the hyperperiod whereas flows 2 and 3 occur three times in the hyperperiod. Notice that we also depicted the feasible routing of flows in Figure 4.2.

Table 4.2: Parameters of the flows from the illustrative example

i	t_i	l_i	$L_{i,a,b}$	r_i	$\tilde{d}_i$	T_i
1	1	5	35	0	150	150
2	2	4	24	0	100	100
3	3	4	24	0	100	100

4.2.4 Scheduling Constraints

The goal of the solver is to instantiate the variables subject to the following constraints.

Routing Constraint

This constraint ensures that each flow is routed through a path from the talker to the listener. To ease the description, we assume that each end-station is connected

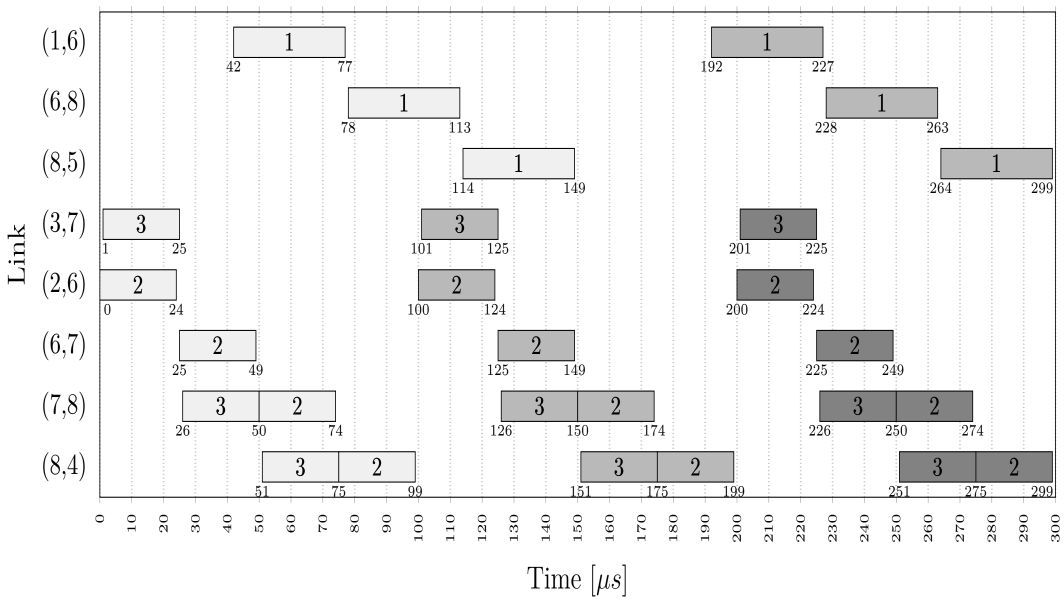


Figure 4.3: Illustration of a schedule with hyperperiod $HP = 300$. Each rectangle represents a frame transmission on the corresponding link, the numbers inside the rectangles are flow IDs, and the numbers by the edges of rectangles are the time points. The first frame occurrences are white, the second occurrences are light gray, and the third occurrences are dark gray.

to exactly one switch. Thus, let (t_i, t'_i) and (l'_i, l_i) denote the first and the last, respectively, link of the routed path of flow i.

$$\begin{aligned}
&\forall i \in \mathcal{F} : \rho_{i,t_i,t'_i} = 1 \\
&\forall i \in \mathcal{F} : \rho_{i,l'_i,l_i} = 1 \\
&\forall i \in \mathcal{F}, \forall a \in \mathcal{V} \setminus \{t_i, l_i\} : \sum_{(x,a)} \rho_{i,x,a} = \sum_{(a,y)} \rho_{i,a,y} \leq 1
\end{aligned} \tag{4.1}$$

The solution to the problem instance from the example in Section 4.2.3 depicted in Figure 4.3 constructed the routing such that $\rho_{1,1,6} = \rho_{1,6,8} = \rho_{1,8,5} = \rho_{2,2,6} = \rho_{2,6,7} = \rho_{2,7,8} = \rho_{2,8,4} = \rho_{3,3,7} = \rho_{3,7,8} = \rho_{3,8,4} = 1$ and all other routing variables are set to 0, which satisfies the routing constraints.

Release and Deadline Constraint

This constraint ensures that transmitting a frame on the first link of the routed path cannot start before the release date and must be completed on the last link before the deadline. Let (t_i, t'_i) and (l'_i, l_i) denote the first and the last, respectively, link of the routed path of flow i.

$$\begin{aligned}
&\forall i \in \mathcal{F} : \phi_{i,t_i,t'_i} \geq r_i \\
&\forall i \in \mathcal{F} : \phi_{i,l'_i,l_i} \leq \tilde{d}_i - L_{i,l'_i,l_i} - d_{l'_i,l_i}
\end{aligned} \tag{4.2}$$

Note that the relative release times and deadlines given by flow parameters r_i and $\tilde{d}_i$ are for the first frame occurrence of the flow. The release times and deadlines of other occurrences are deduced by adding a corresponding multiple of the flow period T_i. However, thanks to the zero-jitter requirement, the satisfaction of release time and deadline for the first frame occurrence implies satisfaction for all frame occurrences.

The solution from Figure 4.3 satisfies all release and deadline constraints. For example, for flow 2, it holds that $0 = \phi_{2,2,6} \geq r_2 = 0$ and $75 = \phi_{2,8,4} \leq \tilde{d}_2 - L_{2,8,4} - d_{8,4} = 100 - 24 - 0.17 = 75.83$.

Resource Constraint

Any two transmissions on a link are not allowed to overlap. More precisely, for each pair of distinct frames on the same link, the transmission of one frame must finish before the transmission of the other frame can start, or vice versa. This constraint must consider all frame occurrences in the hyperperiod. Thanks to the zero-jitter requirement, the transmission offset of the k-th frame occurrence of flow $i \in \mathcal{F}$ on link $(a, b) \in \mathcal{E}$ is equal to $\phi_{i,a,b} + (k-1) \cdot T_i$. With $\wedge$ representing the logical conjunction and $\vee$ representing the logical disjunction, the resource constraint can be written as follows.

$$\begin{gathered}\forall i, j \in \mathcal{F}, i < j, \forall (a,b) \in \mathcal{E}, \\ \forall \alpha \in \{0, \dots, \frac{\mathrm{lcm}\{T_i, T_j\}}{T_i} - 1\}, \forall \beta \in \{0, \dots, \frac{\mathrm{lcm}\{T_i, T_j\}}{T_j} - 1\} : \\ \Big(\rho_{i,a,b} = 1 \ \wedge \ \rho_{j,a,b} = 1\Big) \Rightarrow \\ \Big(\phi_{i,a,b} + \alpha \cdot T_i \geq \phi_{j,a,b} + \beta \cdot T_j + L_{j,a,b}\Big) \ \vee \ \Big(\phi_{j,a,b} + \beta \cdot T_j \geq \phi_{i,a,b} + \alpha \cdot T_i + L_{i,a,b}\Big)\end{gathered} \tag{4.3}$$

In the schedule from Figure 4.3, it can be easily seen from the Gantt chart that all the resource constraints are satisfied since there are no overlapping rectangles. We need to check all the occurrences of flows 2 and 3 on links $(7, 8)$ and $(8, 4)$. For illustration, the transmission of the first frame occurrence of flow 2 on link $(7, 8)$ starts at time $\phi_{2,7,8} + 0 \cdot T_2 = 50 + 0 \cdot 100 = 50$, which is right after the transmission of the first frame occurrence of flow 3 on link $(7, 8)$ is completed, which is at time $\phi_{3,7,8} + 0 \cdot T_3 + L_{3,7,8} = 26 + 0 \cdot 100 + 24 = 50$.

Precedence Constraint

This constraint models the precedence relations, that is, a frame can be transmitted from a switch only after the frame is fully delivered to the switch and processed. Note that the implication is added for each pair of links through which the path can be consecutively routed.

$$\begin{gathered}\forall i \in \mathcal{F}, \forall (x,a), (a,b) \in \mathcal{E} : \\ \Big(\rho_{i,x,a} = 1 \ \wedge \ \rho_{i,a,b} = 1\Big) \Rightarrow \Big(\phi_{i,a,b} \geq \phi_{i,x,a} + L_{i,x,a} + d_{x,a} + d_a + \delta\Big)\end{gathered} \tag{4.4}$$

The schedule from Figure 4.3 satisfies all precedence constraints. For example, for flow 1, it holds that $78 = \phi_{1,6,8} \geq \phi_{1,1,6} + L_{1,1,6} + d_{1,6} + d_6 + \delta = 42 + 35 + 0.17 + 0.5 + 0.1 = 77.77$.

Frame Isolation Constraint

If a frame is lost, another frame may be dispatched from the queue, which can cause delays in the delivery times of other frames, as elaborated in [62] or [8]. To avoid this situation, Craciunas et al. [62] proposed adding the frame isolation constraint. It isolates two different frames such that one frame can arrive in a shared queue only after the other frame is dispatched from the queue.

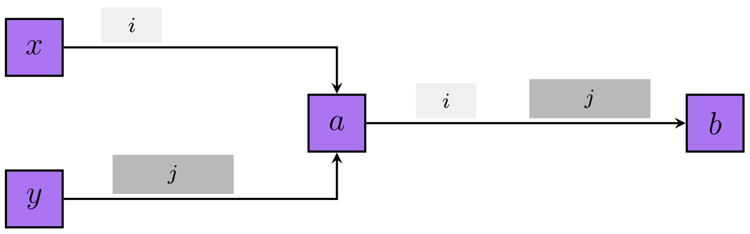


Figure 4.4: Illustration for the frame isolation constraint.

The disjunction on the right-hand side of the implication consists of three disjuncts (see Figure 4.4). The first one states that the $(\alpha+1)$-th frame occurrence of flow i on link (a, b) is dispatched from the switch before the $(\beta + 1)$-th frame occurrence of flow j on link (y, a) is delivered to switch a, the second one states that the $(\beta + 1)$-th frame occurrence of flow j on link (a, b) is dispatched before the $(\alpha + 1)$-th frame occurrence of flow i on link (x, a) is delivered to switch a, and the third one states that the frames are saved in different queues.

$$\begin{aligned}
&\forall i, j \in \mathcal{F}, i < j, \forall (a, b), (x, a), (y, a) \in \mathcal{E},\\
&\qquad \forall \alpha \in \{0, \ldots, \frac{\mathrm{lcm}\{T_i, T_j\}}{T_i} - 1\}, \forall \beta \in \{0, \ldots, \frac{\mathrm{lcm}\{T_i, T_j\}}{T_j} - 1\} :\\
&\qquad\qquad \Big(\rho_{i,a,b} = 1 \ \wedge\ \rho_{j,a,b} = 1 \ \wedge\ \rho_{i,x,a} = 1 \ \wedge\ \rho_{j,y,a} = 1\Big) \Rightarrow\\
&\qquad\qquad \Big(\Big(\phi_{i,a,b} + \alpha \cdot T_i \leq \phi_{j,y,a} + \beta \cdot T_j + L_{j,y,a} + d_{y,a} - \delta\Big) \ \vee\\
&\qquad\qquad \Big(\phi_{j,a,b} + \beta \cdot T_j \leq \phi_{i,x,a} + \alpha \cdot T_i + L_{i,x,a} + d_{x,a} - \delta\Big) \ \vee\\
&\qquad\qquad\qquad\qquad\qquad\qquad \Big(\lambda_{i,a,b} \neq \lambda_{j,a,b}\Big)\Big) \qquad (4.5)
\end{aligned}$$

Note that in contrast to [62], we assume that the frame is saved in a queue after it is entirely received by the switch, which is why the constraint includes the transmission duration.

One can verify that the schedule from Figure 4.3 satisfies the frame isolation constraint. For illustration, let us take a look again on the first frame occurrences of flows 2 and 3 on link $(7, 8)$. The frame isolation constraint is satisfied since the first frame occurrence of flow 3 on link $(7, 8)$ is first dispatched from the queue and then the first frame occurrence of flow 2 on link $(6, 7)$ is stored to the queue. That is, $26 = 26 + 0 \cdot 100 = \phi_{3,7,8} + 0 \cdot T_3 \leq \phi_{2,6,7} + 0 \cdot T_2 + L_{2,6,7} + d_{6,7} - \delta = 25 + 0 \cdot 100 + 24 + 0.17 - 0.1 = 49.07$.

Queue Usage Bounds

This constraint ensures a correct assignment of a queue on each egress port for each frame.

$$\begin{aligned}
&\forall i \in \mathcal{F}, \forall (a,b) \in \mathcal{E} : \rho_{i,a,b} = 1 \Rightarrow (\lambda_{i,a,b} \geq 1)\\
&\forall i \in \mathcal{F}, \forall (a,b) \in \mathcal{E} : \rho_{i,a,b} = 1 \Rightarrow (\lambda_{i,a,b} \leq q_{a,b})
\end{aligned} \tag{4.6}$$

We are indexing the queues from 1 to $q_{a,b}$ instead of from 0 to $q_{a,b} - 1$. This helps to realize the objective function as we can thus distinguish egress ports where no frame is transmitted (and thus no queue is used) from the egress ports where at least one frame is transmitted (and thus at least one queue is used).

In the example, the number of available queues on each link is $q_{a,b} = 1$, hence, all variables $\lambda_{i,a,b}$, for which $\rho_{i,a,b} = 1$, are assigned to 1 and the queue usage bounds are satisfied.

Objective

The objective function that we consider is to minimize the accrued sum of the number of queues used per egress port. The motivation is to keep as many queues as possible for other classes of traffic. As explained in [62], the other types of traffic may then profit from a greater number of queues in the post-analysis, for which techniques like network calculus [85, 86] or trajectory approach [87] are employed. The latency bounds of non-scheduled flows can be greater with the decreasing number of available queues. Hence, reducing the number of queues used by the TT traffic can enhance the timeliness properties and flexibility for the non-scheduled traffic.

Another objective often considered in practice is the minimization of the maximum end-to-end delay related to the flow period. As motivated by [88], this objective, sometimes referred to as response time, is a prevalent objective in TT scheduling, especially on TTEthernet. In our case, however, the end-to-end delays are handled through release times and deadlines, which impose more stringent requirements on the time properties. That is why we opt for optimizing the number of used queues, which appears to have a more considerable motivation in TSN networks.

Formally, let us first introduce auxiliary integer variables $\kappa_{a,b}$, $\forall (a,b) \in \mathcal{E}$, representing the number of queues used on link $(a,b) \in \mathcal{E}$, thus, $\kappa_{a,b} \in \{0, \dots, q_{a,b}\}$. The objective is then realized as follows.

$$\begin{gathered}
\min \sum_{(a,b)\in\mathcal{E}} \kappa_{a,b}\\
\forall i \in \mathcal{F}, \forall (a,b) \in \mathcal{E} : (\rho_{i,a,b} = 1) \Rightarrow (\lambda_{i,a,b} \leq \kappa_{a,b})
\end{gathered} \tag{4.7}$$

Notice that if there is no flow routed through link (a,b), then $\kappa_{a,b} = 0$, otherwise $\kappa_{a,b} \geq 1$. In the example, each link from the Gantt chart in Figure 4.3 uses one queue for TT traffic, whereas the links in the opposite direction (such as $(8,6)$) do not use any queue for TT traffic (thus $\kappa_{8,6} = 0$). Hence, the total number of used queues is 8.

It can be seen that if flow 2 was routed through its shortest path, i.e., links $(2,6)$, $(6,8)$, and $(8,4)$, one queue that is used on link $(6,7)$ would be saved.

However, this routing cannot lead to a feasible solution, i.e., any exact scheduler, given this routing, would report "infeasible". It can be verified that in this case, the resource constraint on link $(6, 8)$ would be violated as the flows 1 and 2 would overlap. Indeed, consider that flow 1 is transmitted at its latest possible time so that the transmission of flow 1 on link $(6, 8)$ starts at 78 as in the figure and that flow 2 is transmitted at its earliest possible time so that the transmission of flow 2 on link $(6, 8)$ starts at 25 (just as on link $(6, 7)$ in the figure). But then the transmission of the second occurrence of flow 1 starts at time $\phi_{1,6,8} + 1 \cdot T_1 = 78 + 1 \cdot 150 = 228$, while the transmission of the third occurrence of flow 2 starts at $\phi_{2,6,8} + 2 \cdot T_2 = 25 + 2 \cdot 100 = 225$ and is completed at 249, thereby causing the overlap. To avoid this overlap, we can try sending flow 1 earlier so that the transmission of second occurrence of flow 1 is completed at time 225, that is, starts at time 190, but then the first occurrence of flow 1 starts at time 40, while the first occurrence of flow 2 is completed at time 49, thereby causing the overlap again. We can also try, oppositely, to send flow 1 as early as possible and flow 2 as late as possible. But then the resource constraint is violated directly by the first occurrences of the flows since the transmission of the first occurrence of flow 1 starts at time 36 and completes at time 71, while the transmission of the first occurrence of flow 2 starts at time 50 and completes at time 74, yielding the overlap again.

The example shows that routing the flows through their shortest paths may prevent finding a feasible solution. There have been attempts in the literature to find an alternative criterion for the routing to increase the chance of finding a feasible solution, e.g., minimizing the maximum scheduled traffic load [63, 72]. However, this criterion would route flow 2 also through its shortest path as we will explain later in Section 4.3.6. Hence, whether the flows are routed through their shortest path or using some other more involved criterion, it may always happen that it does not yield a feasible solution. Notice that this phenomenon happens even in our simple exemplary problem instance that is special in that $r_i = 0$ and $\tilde{d_i} = T_i$, for each flow i. This confirms the motivation to study the JRaS problem.

4.3 Solution

In this section, we explain how we model JRaS in CP. Thanks to particular constructs in CP (see Section 1.4), we propose two models to handle the frame isolation constraint: one with simple disjunctions and one modeling waiting of frames in queues. Then we describe further improvements applied to the models (4.3.3), an approach based on restricting the problem (4.3.4), and, most importantly, the Logic-Based Benders Decomposition (4.3.5) along with various routing criteria (4.3.6).

We start with the description that is valid for both proposed CP models. First, instead of variables $\phi_{i,a,b}$ and $\rho_{i,a,b}$, we introduce optional interval variable $I_{i,k}^{a,b}$ for each frame occurrence, i.e., $\forall i \in \mathcal{F}, \forall (a, b) \in \mathcal{E}, \forall k \in \{0, ..., \frac{HP}{T_i} - 1\}$, $I_{i,k}^{a,b}$ represents $(k + 1)$-th frame occurrence of flow i on link (a, b).

First, we need to ensure the zero jitter. Constraint $\text{StartAtStart}(I_1, I_2, t)$ enforces that the difference between the start times of interval variable I_2 and interval variable I_1 is exactly t time units, but if any of the interval variables is absent, the constraint is implicitly satisfied. Hence, we add:

$$\forall i \in \mathcal{F}, \forall (a,b) \in \mathcal{E}, \forall k \in \{1, ..., \frac{HP}{T_i} - 1\} : \text{StartAtStart}(I^{a,b}_{i,k-1}, I^{a,b}_{i,k}, T_i) \quad (4.8)$$

Also, we need to ensure that all the frame occurrences in the hyperperiod of the same flow are either present or absent. Thus, we add:

$$\forall i \in \mathcal{F}, \forall (a,b) \in \mathcal{E}, \forall k \in \{1, ..., \frac{HP}{T_i} - 1\} : \\ \text{PresenceOf}(I^{a,b}_{i,k-1}) \Leftrightarrow \text{PresenceOf}(I^{a,b}_{i,k}) \quad (4.9)$$

Constraint (4.1) is realized using the Alternative constraint. The definition of the Alternative constraint, which gets an interval variable as the first argument and a set of interval variables as the second argument, is as follows. If the interval variable given as the first argument is present, then exactly one interval variable from the set of interval variables given as the second argument is present, and the others are absent. Besides, it ensures that the start times and completion times of the present interval variables are equal. On the contrary, if the interval variable given as the first argument is absent, then all the interval variables given as the second argument are absent too. Thus, to realize constraint (4.1), we first set $I^{t_i,t'_i}_{i,0}$ and $I^{l'_i,l_i}_{i,0}$ to be present. Then, we introduce interval variables $N^-_{i,a}$ and $N^+_{i,a}$ and add the following constraints:

$$\forall i \in \mathcal{F}, \forall a \in \mathcal{V} \setminus \{t_i, l_i\} :$$

$$\text{Alternative}\left(N^-_{i,a}, \bigcup_{(x,a) \in E} I^{x,a}_{i,0}\right) \quad (4.10)$$

$$\text{Alternative}\left(N^+_{i,a}, \bigcup_{(a,y) \in E} I^{a,y}_{i,0}\right) \quad (4.11)$$

$$\text{PresenceOf}(N^-_{i,a}) \Leftrightarrow \text{PresenceOf}(N^+_{i,a}) \quad (4.12)$$

Constraint (4.2) is done simply by setting the lower bound and upper bound of the domains when declaring the interval variables. The lengths of the interval variables are also set at the declaration.

Constraint (4.3) is realized using the NoOverlap constraint:

$$\forall (a,b) \in \mathcal{E} : \text{NoOverlap}\left(\bigcup_{\substack{i \in \mathcal{F} \\ k \in \{0, ..., \frac{HP}{T_i} - 1\}}} \{I^{a,b}_{i,k}\} \right) \quad (4.13)$$

Constraint (4.4) is realized using constraint StartBeforeStart(I_1, I_2, t), which ensures that the difference between the start times of interval variable I_2 and interval variable I_1 is at least t time units. However, if either of the interval variables is absent, the constraint is implicitly satisfied. This is done for each pair of consecutive links through which a flow can be routed.

$$\forall i \in \mathcal{F}, \forall (x,a), (a,b) \in \mathcal{E} : \text{StartBeforeStart}(I^{x,a}_{i,0}, I^{a,b}_{i,0}, L_{i,x,a} + d_{x,a} + d_a + \delta) \quad (4.14)$$

Constraint (4.6) and objective (4.7) are already in the shape that can be passed to the CP solver.

4.3.1 CP1: Model with Disjunctions

The first way to handle constraint (4.5) is in the form of disjunctions, using multiplication with the predicate PresenceOf as follows:

$$\forall i,j \in \mathcal{F}, i<j, \forall (a,b) \in \mathcal{E}, \forall \alpha \in \Big\{0,\ldots,\frac{\operatorname{lcm}\{T_i,T_j\}}{T_i}-1\Big\}, \forall \beta \in \Big\{0,\ldots,\frac{\operatorname{lcm}\{T_i,T_j\}}{T_j}-1\Big\}:$$
$$\Big(\text{PresenceOf}(I^{a,b}_{i,0}) \ \wedge\ \text{PresenceOf}(I^{a,b}_{j,0}) \ \wedge\ \lambda_{i,a,b}=\lambda_{j,a,b}\Big) \Rightarrow$$
$$\Big(\big(\text{StartOf}(I^{a,b}_{i,0}) \le \sum_{(y,a)\in\mathcal{E}} \text{PresenceOf}(I^{y,a}_{j,0}) \cdot (\text{StartOf}(I^{y,a}_{j,0}) + L_{j,y,a} + d_{y,a} + \beta\cdot T_j - \alpha\cdot T_i - \delta)\big)$$
$$\vee\ \big(\text{StartOf}(I^{a,b}_{j,0}) \le \sum_{(x,a)\in\mathcal{E}} \text{PresenceOf}(I^{x,a}_{i,0}) \cdot (\text{StartOf}(I^{x,a}_{i,0}) + L_{i,x,a} + d_{x,a} + \alpha\cdot T_i - \beta\cdot T_j - \delta)\big)\Big) \tag{4.15}$$

We refer to this model as CP1.

4.3.2 CP2: Model with Interval Variables

Another option to handle constraint (4.5) is to model waiting of frames in queues by interval variables. More precisely, we introduce extra interval variables representing the time intervals when the switch is precluded from receiving and storing frames to a queue. If a frame arrived during this time interval, two frames would share the queue, thereby violating the frame isolation constraint. Imposing NoOverlap constraints over these interval variables, as will be done by equation (4.23), will ensure the isolation of frames.

We introduce interval variables $W^{x,a,b}_i$ representing the time interval since the frame occurrence represented by $I^{x,a}_{i,0}$ is enqueued at switch a until it is transmitted on egress port (a,b). As we do not know the routing of the flows in advance, $W^{x,a,b}_i$ is optional, and we add the following constraints to ensure the correct timing and selection according to the routing:

$$\forall i \in \mathcal{F}, \forall (x,a),(a,b) \in \mathcal{E}:$$
$$\text{StartAtStart}(I^{x,a}_{i,0}, W^{x,a,b}_i, L_{i,x,a}+d_{x,a}-\delta) \tag{4.16}$$
$$\text{EndAtStart}(W^{x,a,b}_i, I^{a,b}_{i,0}, 0) \tag{4.17}$$
$$\text{PresenceOf}(W^{x,a,b}_i) \Leftrightarrow (\text{PresenceOf}(I^{x,a}_{i,0}) \wedge \text{PresenceOf}(I^{a,b}_{i,0})) \tag{4.18}$$

Further, we do not know to which queue the frame will be enqueued. And since the flows may be separated by being assigned to distinct queues, we introduce optional interval variables $\overline{W}^{x,a,b}_{i,q,k}$ representing the same time interval $W^{x,a,b}_i$ but with the distinction that the $(k+1)$-th frame occurrence of flow i going through links (x,a) and (a,b) is stored in queue q. The correct selection of the queue is made by Alternative constraint:

$$\forall i \in \mathcal{F}, \forall (x,a),(a,b) \in \mathcal{E} : \text{Alternative}\Big(W^{x,a,b}_i, \bigcup_{q\in\{1,\ldots,q_{a,b}\}} \overline{W}^{x,a,b}_{i,q,0}\Big) \tag{4.19}$$

The zero jitter and correct presence of all the occurrences is ensured in the same way as for interval variables $I^{a,b}_{i,k}$, and also we must ensure the equal length for all the occurrences:

$$\forall i \in \mathcal{F}, \forall (x,a),(a,b) \in \mathcal{E}, \forall k \in \{1,...,\frac{HP}{T_i}-1\}, \forall q \in \{1,\dots,q_{a,b}\}:$$

$$\text{PresenceOf}(\overline{W}^{x,a,b}_{i,q,k-1}) \Leftrightarrow \text{PresenceOf}(\overline{W}^{x,a,b}_{i,q,k}) \tag{4.20}$$

$$\text{LengthOf}(\overline{W}^{x,a,b}_{i,q,k-1}) = \text{LengthOf}(\overline{W}^{x,a,b}_{i,q,k}) \tag{4.21}$$

$$\text{StartAtStart}(\overline{W}^{x,a,b}_{i,q,k-1}, \overline{W}^{x,a,b}_{i,q,k}, T_i) \tag{4.22}$$

Constraint (4.5) is realized by adding suitable NoOverlap constraints:

$$\forall (a,b) \in \mathcal{E}, \forall q \in \{1,\dots,q_{a,b}\} : \text{NoOverlap}\left(\bigcup_{\substack{(x,a)\in\mathcal{E} \\ i\in\mathcal{F} \\ k\in\{0,\dots,\frac{HP}{T_i}-1\}}} \overline{W}^{x,a,b}_{i,q,k} \right) \tag{4.23}$$

Last but not least, the selection of optional interval variables must correspond to the correct assignment of queues to frames:

$$\forall i \in \mathcal{F}, \forall (a,b) \in \mathcal{E}:$$

$$\text{PresenceOf}(I^{a,b}_{i,0}) \Rightarrow \lambda_{i,a,b} = \sum_{\substack{(x,a)\in\mathcal{E} \\ q\in\{1,\dots,q_{a,b}\}}} q \cdot \text{PresenceOf}(\overline{W}^{x,a,b}_{i,q,0}) \tag{4.24}$$

We refer to this model as CP2.

4.3.3 Basic Implementation Improvements

It is easy to notice that a lot of variables and constraints are added unnecessarily. For example, links outgoing from the listener and links incoming back to the talker can never be selected in a feasible solution. Thus, we preprocess routes such that we enumerate all potential paths from the talker to the listener for each flow and introduce variables $I^{a,b}_{i,k}$ and $\lambda_{i,a,b}$ only if there exists a path for flow $i \in \mathcal{F}$ going from t_i to l_i through link $(a,b) \in \mathcal{E}$. Although the number of paths in a general graph is known to grow exponentially, the time consumed by the preprocessing phase in our case is not an issue.

Furthermore, we calculate, $\forall i \in \mathcal{F}, \forall (a,b) \in \mathcal{E}$, the earliest possible transmission offset $est_{i,a,b}$ and the latest possible transmission offset $lst_{i,a,b}$, which is done as follows. All values $est_{i,a,b}$ are initially set to ∞ and $lst_{i,a,b}$ are set to $-\infty$. Then, for each flow and each enumerated path, we first check whether the path is short enough so that the flow can be delivered through the path respecting its release date and deadline but disregarding all other flows in the problem instance. If yes, the values of $est_{i,a,b}$ are updated in the order of links in the path starting from the talker, $prec(a)$ denoting the preceding node of a in the path:

$$est_{i,t_i,t'_i} \leftarrow r_i \tag{4.25}$$

$$est_{i,a,b} \leftarrow \min\{est_{i,a,b}, est_{i,prec(a),a} + L_{i,prec(a),a} + d_{prec(a),a} + d_a + \delta\} \tag{4.26}$$

Notice that the min function is used as more alternative paths can go through the same link, then we need to take the smallest value of $est_{i,a,b}$ from these paths.

Analogically, the values of $lst_{i,a,b}$ are updated in the reversed order of links in the path starting from the listener, $succ(a)$ denoting the succeeding node of a in the path:

$$lst_{i,l'_i,l_i} \leftarrow \tilde{d}_i - L_{i,l'_i,l_i} - d_{l'_i,l_i} \tag{4.27}$$

$$lst_{i,a,b} \leftarrow \max\{lst_{i,a,b}, lst_{i,b,succ(b)} - L_{i,a,b} - d_{a,b} - d_b - \delta\} \tag{4.28}$$

After the bounds are calculated, it is clear that if $est_{i,a,b} > lst_{i,a,b}$ then there is no feasible path of flow i going through link (a, b). Hence, we introduce the corresponding variables if and only if $est_{i,a,b} \leq lst_{i,a,b}$. Besides, we tighten the bounds of the interval variables such that the minimum and the maximum start time are at least $est_{i,a,b}$ and at most $lst_{i,a,b}$, respectively.

4.3.4 Restricting Queues to one

Recall that the models just described are minimizing the overall sum of used queues. Another option to tackle the problem is to solve the restricted variant such that the number of queues for each egress port is confined to 1. The search space thus becomes much smaller. This way, some problem instances can be deemed infeasible that would be solved with more than one queue on some egress ports. On the other hand, we do not need variables $\lambda_{i,a,b}$, and the constraints in the models thus simplify significantly. More precisely, constraint (4.6), objective (4.7), and the last disjunct from constraint (4.5) are omitted.

This approach will be referred to as with one queue and labeled by 1Q, whereas the approach optimizing the number of queues used will be labeled as OPT.

4.3.5 Logic-Based Benders Decomposition

In the models described so far, the routes of the flows and transmission offsets of frames are to be found simultaneously. Here we propose to decompose the problem in two phases: first, compute the routes of the flows, that is, assign values to variables $\rho_{i,a,b}$, and second, schedule the routes, that is, assign values to variables $\phi_{i,a,b}$ and $\lambda_{i,a,b}$. After the second phase, go back to the first phase to find another routing and repeat. In general, this technique is referred to as Logic-Based Benders Decomposition and has been widely used on various combinatorial optimization problems (e.g., [89]). The scheme applied to JRaS is illustrated in Figure 4.5.

More precisely, we first build a model for the routing problem consisting of variables $\rho_{i,a,b}$, Constraints (4.1), and a criterion function that will be described further. Note that to distinguish the objective function in the routing problem from the objective function of JRaS, which is the minimization of the number of used queues, we call the objective function in the routing problem a criterion function.

If the routing model reports "infeasible", there is no solution to the problem. If it finds a feasible routing, we construct a scheduling subproblem based on the routing. The scheduling subproblem is composed of variables $\phi_{i,a,b}$ and $\lambda_{i,a,b}$, and Constraints (4.2)–(4.7). However, as we already know the values of all $\rho_{i,a,b}$, we add only the constraints that are not trivially valid, i.e., a constraint is added when all the routing variables on the left-hand side of the implication are true.

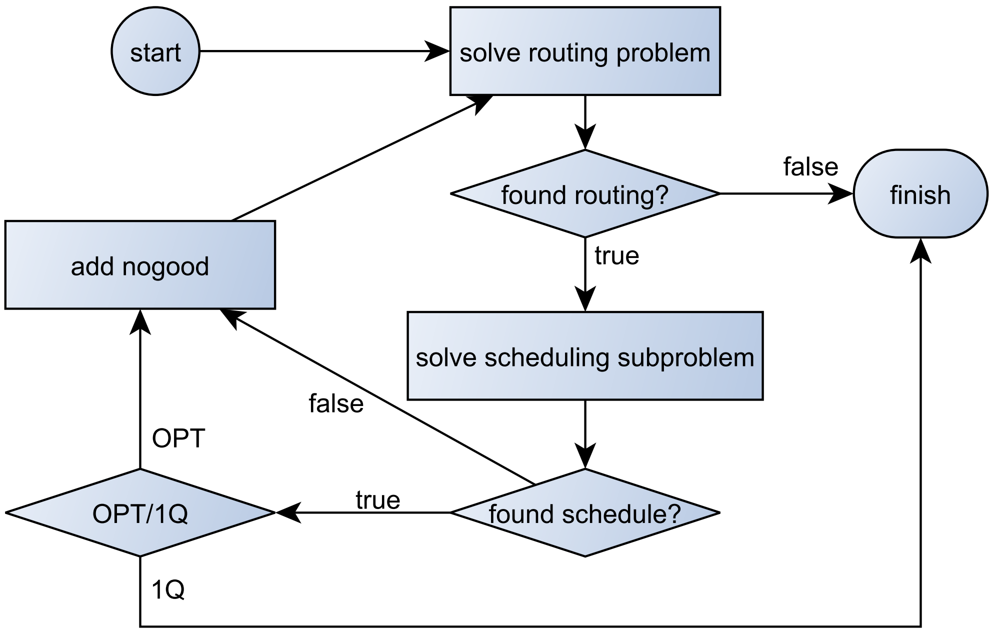


Figure 4.5: Workflow of the Logic-Based Benders Decomposition.

For our CP models, it means, for example, that only interval variables $I_{i,k}^{a,b}$ are created for which $\rho_{i,a,b} = true$, and these interval variables are no longer optional, but present.

When the run of the scheduling subproblem is finished, regardless of whether it found a feasible schedule or not, we proceed to try another routing. To make sure the new routing has not yet been tried, we add a *nogood* (a new constraint) to the routing problem model. If the scheduling subproblem found a feasible schedule, we add a nogood such that the new solution must be different. If the scheduling subproblem is infeasible, we try to retrieve a set of constraints that cannot be satisfied together and build the nogood such that some variable involved in the set of constraints must be changed. After adding the nogood to the routing problem, the solver is invoked again. This process is repeated until the time limit is reached or there is no other routing to try. Finally, the best-found solution is reported.

To illustrate how the nogood is created, let us consider the example from Section 4.2.3 and assume all flows are routed through their shortest paths. After the run of the scheduling subproblem, suppose (and it is strictly hypothetical just for the sake of explanation) the set of constraints that cannot be satisfied together looked as follows:

$$\begin{gathered}
\phi_{1,1,6} \geq r_1 \\
\phi_{1,8,5} \leq \tilde{d}_1 - L_{1,8,5} - d_{8,5} \\
\phi_{2,6,8} \geq \phi_{2,2,6} + L_{2,2,6} + d_{2,6} + d_6 + \delta \\
(\phi_{1,6,8} + 1 \cdot T_1 \geq \phi_{2,6,8} + 2 \cdot T_2 + L_{2,6,8}) \ \vee \ (\phi_{2,6,8} + 2 \cdot T_2 \geq \phi_{1,6,8} + 1 \cdot T_1 + L_{1,6,8})
\end{gathered} \tag{4.29}$$

Then the nogood would be constructed as follows:

$$\rho_{1,1,6} + \rho_{1,8,5} + \rho_{2,6,8} + \rho_{2,2,6} + \rho_{1,6,8} \leq 4 \tag{4.30}$$

To find the set of constraints that cannot be satisfied together, we use the CP method called *conflict refiner*. It directly finds the set of variables that cannot be consistently instantiated altogether. Note that the set of constraints that cannot be satisfied together is in SMT referred to as *unsatisfiable core* and in ILP as *irreducible inconsistent subsystem*. Most solvers usually provide a method to retrieve this set.

In the experiments, to solve the scheduling subproblem, we evaluated both models CP1 and CP2, as well as all the baseline approaches. The routing problem, however, is solved by ILP, but it could be equally solved by any other method as the runtime for the routing problem is negligible compared to the scheduling subproblem.

For the 1Q approach, the algorithm terminates as soon as it finds the first feasible schedule. For the OPT approach, the objective value of the best incumbent solution can be used in the scheduling subproblem such that we enforce that the objective value of the new solution to the scheduling subproblem can only be better than that of the best incumbent solution. As confirmed by the preliminary results, adding an extra constraint to enforce the objective value to improve the best incumbent solution is beneficial for SMT but slightly detrimental for ILP and CP. Hence, we did not include this extra constraint in the experimental evaluation of CP. In ILP, however, the solver provides an extra parameter called *cutoff*, which states that we are interested only in solutions not worse than the value of cutoff, and this value is efficiently used by the solver.

To distinguish this approach from the global models, it is in what follows referred to as decomposed model or decomposition and labeled with suffix D, whereas the global approaches are labeled with suffix G. Note that the decomposition does not prune out any solution as all the possible routings would be tried if given enough time. Hence, if given infinite time, the results of the global and decomposed approaches would be equal.

4.3.6 Routing Problem Criteria

We investigate three fundamental criterion functions for the routing problem. The importance of the routing and its impact on the schedulability and the computation time was studied in [90].

Shortest Paths (SP)

The traditional criterion for the routing is minimizing the *total length of paths* in the number of hops, that is:

$$\min \sum_{\substack{i \in \mathcal{F} \\ (a,b) \in \mathcal{E}}} \rho_{i,a,b} \tag{4.31}$$

In the first set of experiments, we use SP as the criterion function for the routing problem, whereas the second set of experiments compares various other criteria.

Load Balancing (LB)

Another criterion that appears in the literature is minimizing the *maximum scheduled traffic load* on a link. This criterion was proposed by [63] and [72] and is realized as follows: A continuous variable υ is introduced to represent the maximum utilization of all links by adding the following constraints:

$$\forall(a,b) \in \mathcal{E} : \sum_{i\in\mathcal{F}} \rho_{i,a,b} \cdot \frac{L_{i,a,b}}{T_i} \leq \upsilon \tag{4.32}$$

Then the criterion function is added as such:

$$\min \upsilon \tag{4.33}$$

In our example from Section 4.2.3, this criterion would route flow 2 through its shortest path since the maximum traffic load would be on link $(6,8)$ of value $\frac{L_{1,6,8}}{T_1} + \frac{L_{2,6,8}}{T_2} = \frac{35}{150} + \frac{24}{100} = \frac{142}{300}$, which is lower than if flow 2 is routed through link $(7,8)$, where the traffic load would be of value $\frac{L_{2,7,8}}{T_2} + \frac{L_{3,7,8}}{T_3} = \frac{24}{100} + \frac{24}{100} = \frac{144}{300}$.

Period Matching (PM)

As can be seen in the example in Section 4.2.3, if flows of different periods are routed through the same link, it is unschedulable, whereas the flows of the same periods on the same link can still be scheduled. The motivation is to route through a link such flows whose periods are identical or at least somehow congenial, rather than those flows whose periods are indivisible. We want the criterion function to reflect the extent of the pair-wise *indivisibility* of flows if the flows are routed through the same link. In other words, it is preferable to route through the same link flows of periods 100 and 200, rather than 100 and 150 or 150 and 200.

To assess the extent of the indivisibility, we take the least common multiple (lcm) of the flows divided by their greatest common divisor (gcd). For example, $\frac{\text{lcm}\{100,150\}}{\text{gcd}\{100,150\}} = \frac{300}{50} = 6$ or $\frac{\text{lcm}\{150,200\}}{\text{gcd}\{150,200\}} = \frac{600}{50} = 12$, whereas in the case of harmonic periods, gcd is equal to the shortest period and lcm is equal to the longest period, e.g., $\frac{\text{lcm}\{100,200\}}{\text{gcd}\{100,200\}} = \frac{200}{100} = 2$.

Formally, let us first define the set of distinct periods of the flows as:

$$\mathcal{P} = \{P_1, P_2, \dots, P_{|\mathcal{P}|}\} \tag{4.34}$$

In our example from Section 4.2.3, $\mathcal{P} = \{100, 150\}$, $|\mathcal{P}| = 2$, $P_1 = T_2 = T_3 = 100$, and $P_2 = T_1 = 150$.

Next, we introduce auxiliary variables as follows:

1. $\forall(a,b) \in \mathcal{E}, \forall p \in \{1,\dots,|\mathcal{P}|\}$, binary variable $x_{p,a,b}$, indicating that there is at least one flow of period P_p routed through link (a,b), and

2. $\forall(a,b) \in \mathcal{E}, \forall p,p' \in \{1,\dots,|\mathcal{P}|\}, p < p'$, binary variable $y_{p,p',a,b}$, indicating that there is at least one flow of period P_p and at least one flow of period $P_{p'}$ routed through link (a,b).

First, we need to ensure the correct behavior of these variables. As to variables $x_{p,a,b}$, we add the following constraints:

$$\forall (a,b) \in \mathcal{E}, \forall p \in \{1,\ldots,|\mathcal{P}|\} : x_{p,a,b} \leq \sum_{i \in \mathcal{F}, T_i = P_p} \rho_{i,a,b} \tag{4.35}$$

$$\forall (a,b) \in \mathcal{E}, \forall p \in \{1,\ldots,|\mathcal{P}|\}, \forall i \in \mathcal{F}, T_i = P_p : \rho_{i,a,b} \leq x_{p,a,b} \tag{4.36}$$

And as to variables $y_{p,p',a,b}$, we add the constraints modeling that $y_{p,p',a,b} = 1 \Leftrightarrow x_{p,a,b} = 1 \wedge x_{p',a,b} = 1$, as follows:

$$\forall (a,b) \in \mathcal{E}, \forall p,p' \in \{1,\ldots,|\mathcal{P}|\}, p < p' :$$

$$y_{p,p',a,b} \leq x_{p,a,b} \tag{4.37}$$

$$y_{p,p',a,b} \leq x_{p',a,b} \tag{4.38}$$

$$x_{p,a,b} + x_{p',a,b} \leq y_{p,p',a,b} + 1 \tag{4.39}$$

Finally, the criterion function is constructed as follows:

$$\min \sum_{\substack{(a,b) \in \mathcal{E} \\ p,p' \in \{1,\ldots,|\mathcal{P}|\}, p < p'}} \frac{\operatorname{lcm}\{P_p, P_{p'}\}}{\gcd\{P_p, P_{p'}\}} y_{p,p',a,b} \tag{4.40}$$

In the resulting routing from our example, variables $x_{1,1,6}$, $x_{1,6,8}$, $x_{1,8,5}$, $x_{2,2,6}$, $x_{2,6,7}$, $x_{2,7,8}$, $x_{2,8,4}$, and $x_{2,3,7}$ are 1, the other variables are 0. The criterion function depicted by equation (4.40) is equal to 0, which is clearly optimal.

4.4 Baseline Approaches

Recall that our main contribution is in the CP1 and CP2 models. To compare our work against some baseline approaches, we considered the following state-of-the-art approaches as described in related work:

- The global models proposed by [69] in SMT and ILP and the versions for scheduling a given routing by [62] (in SMT) and [77] (in ILP).
- The no-wait global model in ILP [71] and the no-wait model for scheduling a given routing [73], which we refer to as ILP_NW.
- The time-indexed formulation (also no-wait) in ILP [74], which we refer to as ILP_TI.

Except for ILP_TI, we adjusted these methods to our problem definition both for the OPT approach and the 1Q approach, and we implemented the Logic-Based Benders Decomposition as described in Section 4.3.5. In all the cases, we applied the improvements described in Section 4.3.3.

One might think that imposing the no-wait constraint is a way to circumvent the frame isolation constraint. However, we show that even with the no-wait constraint, the order of frames in a queue is uncertain, thereby losing the deterministic guarantees. Recall we need to find a transmission offset for each frame.

As we work with the granularity of 1 microsecond, the no-wait constraint is realized such that the transmission offset of a frame on a succeeding link is equal to the time point when the frame from the preceding link is ready on the corresponding queue but rounded up to microseconds. More precisely, the precedence constraint is modified as follows:

$$\forall i \in \mathcal{F}, \forall (x,a),(a,b) \in \mathcal{E}: \\ \Big(\rho_{i,x,a} = 1 \ \wedge \ \rho_{i,a,b} = 1\Big) \Rightarrow \Big(\phi_{i,a,b} = \Big\lceil \phi_{i,x,a} + L_{i,x,a} + d_{x,a} + d_a + \delta \Big\rceil\Big) \quad (4.41)$$

Consider two flows in the scenario from Figure 4.4. Let:

$$L_{i,x,a} = L_{i,a,b} = 0.8 \quad (4.42)$$
$$L_{j,y,a} = L_{j,a,b} = 3.85 \quad (4.43)$$
$$d_{x,a} = d_{y,a} = d_{a,b} = 0.17 \quad (4.44)$$
$$d_a = 10 \quad (4.45)$$
$$\delta = 0 \quad (4.46)$$

This tiny problem instance can be scheduled subject to the no-wait constraint as follows:

$$\phi_{i,x,a} = 3 \quad (4.47)$$
$$\phi_{j,y,a} = 0 \quad (4.48)$$
$$\phi_{i,a,b} = \lceil \phi_{i,x,a} + L_{i,x,a} + d_{x,a} + d_a + \delta \rceil = \lceil 13.97 \rceil = 14 \quad (4.49)$$
$$\phi_{j,a,b} = \lceil \phi_{j,y,a} + L_{j,y,a} + d_{y,a} + d_a + \delta \rceil = \lceil 14.02 \rceil = 15 \quad (4.50)$$

It can be easily verified that the frame isolation constraint is not satisfied. Moreover, the times when the frames are enqueued at switch a are:

$$\phi_{i,x,a} + L_{i,x,a} + d_{x,a} = 3.97 \quad (4.51)$$
$$\phi_{j,y,a} + L_{j,y,a} + d_{y,a} = 4.02 \quad (4.52)$$

It follows that if there is maximum synchronization error $\delta > 0.05$, the order of the two frames in a queue is not deterministic. Hence, the network does not guarantee deterministic behavior. For this reason, we enforce the frame isolation constraint in our implementation, even in the no-wait models.

4.5 Experiments

We implemented the models in the IBM CP Optimizer version 12.10 [11]. We set parameter `ConflictRefinerOnVariables` to `On` in order to make the solver find the set of variables that cannot be consistently instantiated altogether. Next, we set the parameter `NoOverlapInferenceLevel` to `Extended` in order to make the solver use improved filtering for the NoOverlap constraints.

To solve the SMT models, we use the application interface of the Z3 solver version 4.8.8 [24]. This solver provides two objects: *Solver*, which serves for determining the feasibility of a formula and is thus used for the 1Q approaches, and *Optimizer*, which allows for optimization of an objective function and is thus used for the OPT approaches. In the *Solver* object, we set `unsat_core` to `true` in order to make the solver find the unsatisfiable core.

To solve the ILP models, we use Gurobi solver version 9.0.1 [14]. The experiments were run on a system with 2x Intel® Xeon® E5-2690 v4 CPU, 14 Cores/CPU; 2.6GHz; 35 MB SmartCache; with 512 GB DDR4, ECC.

4.5.1 Problem Instances

We evaluated our algorithms on randomly generated problem instances of topologies proposed by [63] inspired by industrial automation[1]. The switches are connected either in a ring or as a partial mesh. The small networks consist of 12 switches and 12 end-stations, the large networks consist of 48 switches and 48 end-stations. Each end-station is connected to a different switch. The small partial meshes are of types PM3-4-3 and PM4-3-2, and the large partial meshes are of types PM6-8-3 and PM8-6-2. The numbers 4-3-2 stand for 4 columns, 3 switches per column, and horizontal connections every 2 rows (see Figure 4.6).

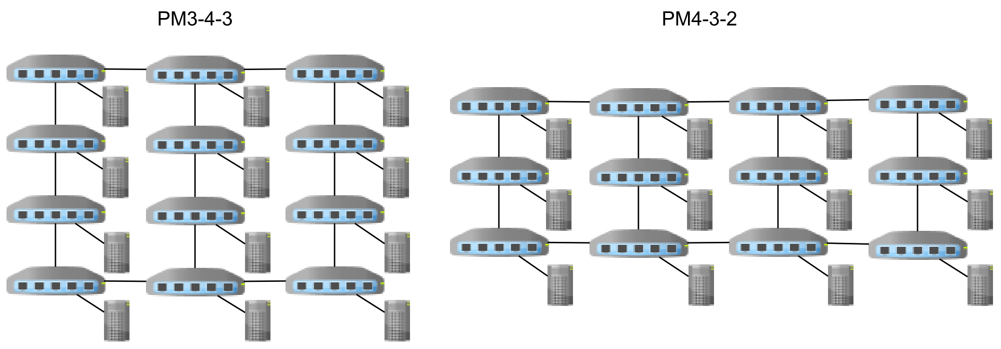


Figure 4.6: Small partial-mesh topologies.

The other parameters were set as follows. $\forall a \in \mathcal{V} : d_a = 10$; $\forall (a,b) \in \mathcal{E} : d_{a,b} = 0.17, q_{a,b} = 2$. The speed of all links was set to 1000 Mbit/s.

Table 4.3: Period Sets

Periods	HP
$\{400, 500, 1000\}$	2000
$\{500, 1000, 2000\}$	2000
$\{800, 1600, 3200\}$	3200
$\{500, 1000, 1500, 3000\}$	3000
$\{500, 800, 1000, 2000, 4000\}$	4000

The number of flows in the small networks ranged from 10 to 300, incremented by 10, and in the large networks ranged from 30 to 900, incremented by 30. The talker and listener were chosen uniformly at random for each flow. The periods T_i were uniformly chosen from each of the period sets listed in Table 4.3.

[1]The problem instances can be downloaded from `https://github.com/CTU-IIG/JRaS-TSN`

The payload of each flow was set uniformly at random from interval $[72, 1542]$ bytes, where 1542 bytes is the maximal Ethernet packet size including the overhead. The release dates and deadlines were generated randomly such that interval $[r_i, \tilde{d}_i]$ covers exactly 50 % of period T_i, which is the parameter we obtained from our industrial partner for applications in industrial automation. For each set of parameters, we generated ten random problem instances for the small networks and one random instance for the large networks. The small networks were given a time limit of 60 seconds, and the large networks were given a time limit of 3600 seconds.

4.5.2 Comparison of Schedulability

First of all, we are interested in the schedulability ratio, which is the percentage of problem instances for which a feasible schedule was found by a given method. Table 4.4 shows the percentage of instances for which a feasible schedule was found by the OPT approaches. Analogically, Table 4.5 shows the results for the 1Q approaches. The column "Any" in the tables shows the percentage of instances for which a feasible schedule was found by at least one (any) of all the approaches from the table. This column is meant to illustrate the extent to which the schedulable instances differ among various approaches and how the winning approach covers the subset of instances scheduled by any approach.

Table 4.4: Percentage of problem instances scheduled by OPT approaches

		Global					Decomposed				
topology	Any	SMT	ILP	ILP_NW	CP1	CP2	SMT	ILP	ILP_NW	CP1	CP2
PM3-4-3	76.5	3.5	12.8	14.7	16.4	50.7	20.9	49.5	50.9	46.7	**75.0**
PM4-3-2	77.8	3.4	11.9	13.1	18.6	48.0	22.8	51.3	53.8	47.3	**76.3**
RING12	70.7	4.9	18.7	19.2	14.5	48.1	16.7	42.7	44.9	37.3	**69.7**
mean	75.0	3.9	14.5	15.7	16.5	49.0	20.2	47.9	49.9	43.8	**73.7**
PM6-8-3	49.3	-	-	-	-	1.3	13.3	25.3	28.7	22.0	**47.3**
PM8-6-2	60.0	-	-	-	-	2.0	12.7	24.7	30.0	21.3	**58.7**
RING48	60.7	-	-	-	-	29.3	13.3	30.7	40.0	22.7	**58.0**
mean	56.7	-	-	-	-	10.9	13.1	26.9	32.9	22.0	**54.7**

Table 4.5: Percentage of problem instances scheduled by 1Q approaches

		Global					Decomposed					
topology	Any	SMT	ILP	ILP_NW	CP1	CP2	SMT	ILP	ILP_NW	ILP_TI	CP1	CP2
PM3-4-3	69.5	6.4	10.8	11.5	38.3	34.7	31.7	43.9	46.7	13.9	**68.2**	44.6
PM4-3-2	71.1	5.9	9.0	9.3	34.8	37.1	33.1	45.6	49.5	14.9	**70.3**	46.9
RING12	63.4	8.6	16.1	18.7	42.0	31.9	29.1	39.3	41.3	12.4	**62.1**	41.5
mean	68.0	7.0	12.0	13.2	38.4	34.6	31.3	43.0	45.8	13.8	**66.8**	44.4
PM6-8-3	36.0	-	-	-	-	6.0	22.0	28.0	**30.7**	-	**30.7**	18.7
PM8-6-2	35.3	-	-	-	-	4.7	21.3	26.7	28.7	-	**34.0**	20.7
RING48	51.3	-	-	-	-	14.0	23.3	34.7	42.0	-	**47.3**	26.0
mean	40.9	-	-	-	-	8.2	22.2	29.8	33.8	-	**37.3**	21.8

It can be seen that our CP methods reach the highest schedulability. In particular, CP2 combined with the decomposition achieved the highest schedulability among the OPT approaches, whereas CP1 combined with the decomposition achieved the highest schedulability among the 1Q approaches. As expected, limiting queues to one renders some instances unsolvable that could be solved using more queues. On the other hand, limiting queues to one increases the schedulability rate for the CP1 and SMT models, which shows that the size of the search space is crucial for the performance of these models. It can also be seen that for every method, the decomposition outperforms the global model.

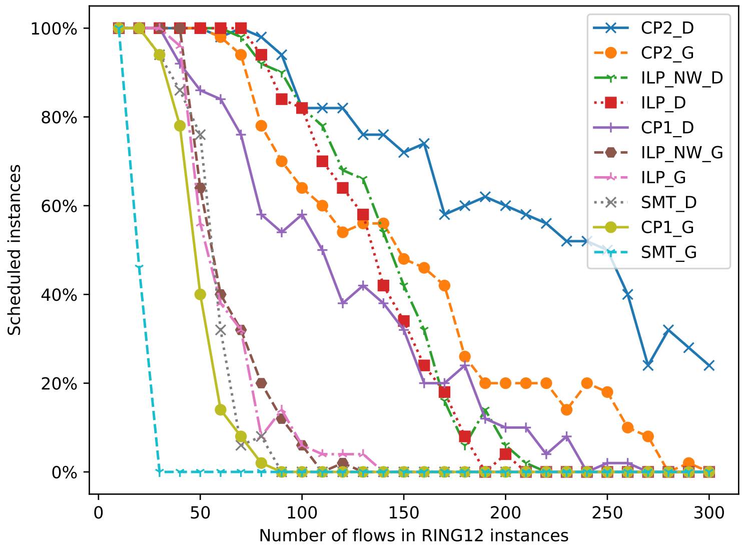


Figure 4.7: Dependence of the percentage of scheduled problem instances on the number of flows in RING12 networks for OPT approaches.

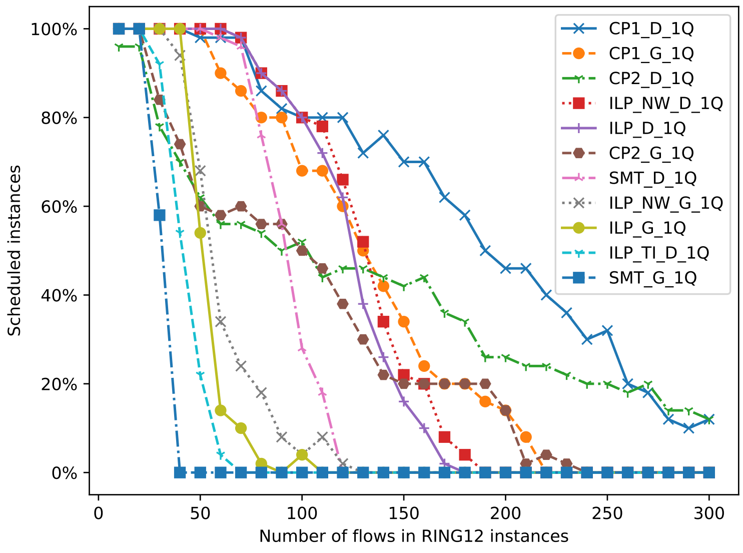


Figure 4.8: Dependence of the percentage of scheduled problem instances on the number of flows in RING12 networks for 1Q approaches.

The dependence of schedulability on the number of flows in the ring networks with 12 switches is depicted in Figure 4.7 and Figure 4.8. Other networks exhibited similar trends. The methods in the diagrams are sorted in the decreasing order of the schedulability. Recall that the global approaches are labeled with suffix G, and the decomposed approaches are labeled with suffix D.

The solvers almost always deplete the given time limit. That is why we focused on whether a feasible solution was found in the given time limit and the quality of such a solution. However, with the increasing complexity of the instances, the global models tend to take more time than the time limit. This is particularly noticeable for SMT, as can be seen in Figure 4.9 and Figure 4.10, where the dependence of the average solving time on the number of flows in the ring networks with 12 switches is shown. Recall that the time limit for the small networks was 60 s; hence, no curve should exceed the value of 60 on the y-axis. We note that the biggest excess of the time-limit was 5635.62 s for the SMT_G on a PM3-4-3 instance with 210 flows, which confirms the inapplicability of SMT. Besides, the SMT optimizer has the drawback that it either finds an optimal solution or finds no solution at all.

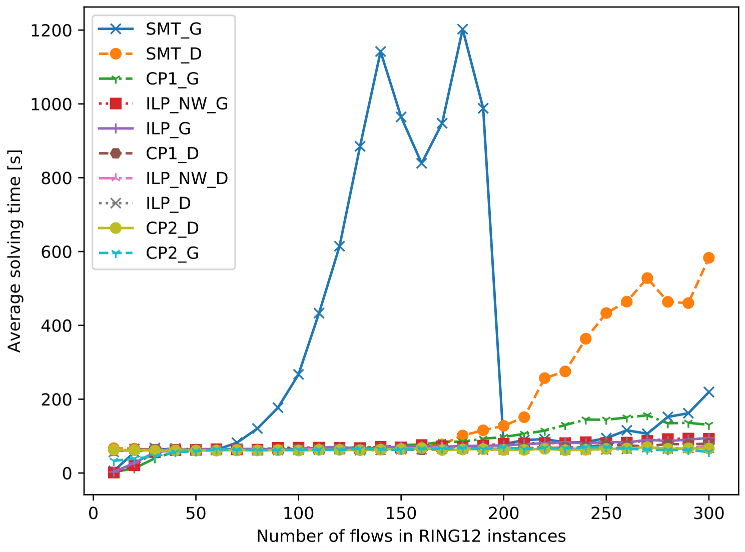


Figure 4.9: Dependence of the average solving time on the number of flows in RING12 networks for OPT approaches.

There are missing results in the tables since the solvers crashed on a lack of memory. The only global model that managed to complete the results for the large networks is CP2, albeit exceeding the time limit. This underlines the importance of decomposition on larger networks.

Overall, it can be seen that our CP models perform best. The results show that CP2 is better at finding a feasible solution in a larger search space, whereas CP1 is better at finding a feasible solution when the search space is pruned by limiting queues to one. We note that we also tried the CP modeling trick known to be efficient for problems where a feasible solution is difficult to find: relaxing deadlines to due dates and involving the violation of the deadlines in the objective function as tardiness. In our case, unfortunately, this brought no benefit. We also tried to set the search phases for the solver such that it schedules first the flows with the smallest periods, then those with the second smallest periods, and so on, just like in the widely used rate monotonic policy [91]. This strategy turned out to be counterproductive since the solver has more sophisticated variable selection

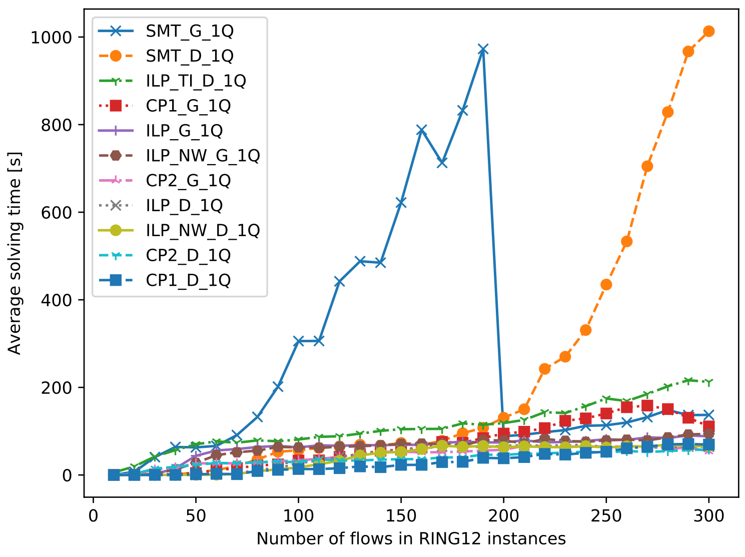


Figure 4.10: Dependence of the average solving time on the number of flows in RING12 networks for 1Q approaches.

rules.

4.5.3 Usage of Queues and Optimality

Table 4.6 shows the percentage of instances for which the models optimizing the number of used queues proved that the found solution is optimal, and Table 4.7 shows the percentage of instances that were proved infeasible. CP1 performed best at proving the optimality, whereas CP2 at proving the infeasibility. Note that the instances are generated such that it is not known whether the unsolved instances are infeasible until some model proves it. Although significantly more successful in finding a feasible solution, the results also confirm that the decomposed models are not suitable for proving the optimality or infeasibility. The reason is that the decomposed models must examine all potential routings (except those pruned out by nogoods), which is rarely made within the time limit.

Table 4.6: Percentage of problem instances with proved optimality

	Global				Decomposed			
topology	SMT	ILP	CP1	CP2	SMT	ILP	CP1	CP2
PM3-4-3	3.53	6.47	**10.33**	0.27	0.00	0.00	0.00	0.00
PM4-3-2	3.40	6.20	**10.60**	0.13	0.00	0.00	0.00	0.00
RING12	4.87	7.67	**10.40**	6.93	0.07	0.53	0.47	0.00
mean	3.93	6.78	**10.44**	2.44	0.02	0.18	0.16	0.00
PM6-8-3	-	-	-	0.00	0.00	0.00	0.00	0.00
PM8-6-2	-	-	-	0.00	0.00	0.00	0.00	0.00
RING48	-	-	-	10.00	0.67	1.33	1.33	0.67
mean	-	-	-	3.33	0.22	0.44	0.44	0.22

Table 4.7: Percentage of problem instances with proved infeasibility

	Global				Decomposed			
topology	SMT	ILP	CP1	CP2	SMT	ILP	CP1	CP2
PM3-4-3	0.00	0.00	0.20	**1.20**	0.00	0.00	0.00	0.00
PM4-3-2	0.00	0.00	0.00	**0.87**	0.00	0.00	0.00	0.00
RING12	0.00	0.00	1.27	**1.40**	0.00	0.00	0.00	0.00
mean	0.00	0.00	0.49	**1.16**	0.00	0.00	0.00	0.00
PM6-8-3	-	-	-	2.00	0.00	0.00	0.00	0.00
PM8-6-2	-	-	-	1.33	0.00	0.00	0.00	0.00
RING48	-	-	-	3.33	0.00	0.00	0.00	0.00
mean	-	-	-	2.22	0.00	0.00	0.00	0.00

Table 4.8 shows the comparison of the usage of queues for CP1 and CP2. Score "x : y" in column "method1 : method2" states that x % of instances were solved by method1 using strictly fewer queues than method2 and that y % of instances were solved by method2 using strictly fewer queues than method1, provided the instance was scheduled by both methods. It can be seen that the schedules found by CP1 are using fewer queues than by CP2, whether it is in the global model (first column) or decomposition (second column). Further, solving the problem by the global model brings an advantage over decomposition for CP1 (third column), whereas the opposite holds for CP2 (fourth column).

Table 4.8: Comparison of better usage of queues

topology	CP1_G : CP2_G	CP1_D : CP2_D	CP1_G : CP1_D	CP2_G : CP2_D
PM3-4-3	6.7 : 3.1	16.1 : 0.4	10.7 : 4.6	10.0 : 39.5
PM4-3-2	9.6 : 2.7	15.9 : 0.4	11.2 : 6.7	8.3 : 38.7
RING12	4.6 : 2.5	11.9 : 0.4	8.9 : 2.7	10.4 : 35.3
mean	7.0 : 2.8	14.6 : 0.4	10.3 : 4.6	9.6 : 37.8
PM6-8-3	- : -	10.0 : 4.0	- : -	0.7 : 0.7
PM8-6-2	- : -	6.0 : 6.7	- : -	0.0 : 2.0
RING48	- : -	4.7 : 3.3	- : -	1.3 : 18.0
mean	- : -	6.9 : 4.7	- : -	0.7 : 6.9

To summarize the results, CP2 is the best in terms of schedulability. In particular, CP2_D should be used if the time limit within which a feasible schedule must be found is small, as this method exhibited the best schedulability ratio. On the other hand, CP1 yields solutions of better quality and is better at proving the optimality of the found solution.

Note that the SMT and ILP models introduce variables only for the first frame occurrences, which is possible due to the zero-jitter requirement. On the contrary, the CP1 and CP2 models work with variables for each frame occurrence, which suggests that if the zero-jitter requirement is relaxed, our CP models should exhibit more efficiency than other models.

4.5.4 Impact of Various Routing Criteria, Period Sets, and Problem Modifications

The following experiments investigate the effect of the different criterion function used for the routing problem in the decomposition. In the experiments above, the decomposition only used the shortest paths as the routing problem criterion, and the scheduling subproblem always got all the remaining time limit. Since the scheduling subproblem creates the bottleneck of the whole problem, it rarely happens that the decomposition examines more than one potential routing. Hence, to focus more on the routing criteria, we modify the decomposition such that the scheduling subproblem gets half of the remaining time limit (but at least one second). This way, we achieve that the decomposition examines several possible routings. Besides, we generated problem instances where the period sets are not particularly congenial (e.g., harmonic periods), as was the case in the experiments so far. The other parameters and the methodology of generating the problem instances are the same. For the comparison, we use only the CP2 model as it performed best.

Table 4.9: Percentage of scheduled problem instances for various routing criteria

Periods	HP	Any	SP	LBSP	SPLB	SPPM	PMSP
$\{1400, 1800, 2400, 3600\}$	50400	25.3	22.3	**23.0**	22.3	22.8	22.0
$\{1400, 1800, 2400\}$	50400	9.1	7.2	7.2	7.6	7.3	**8.3**
$\{1400, 1800, 2800, 3600\}$	25200	39.2	32.3	**33.2**	31.8	32.1	28.3
$\{1400, 1800, 2800\}$	25200	26.3	20.6	20.3	20.3	**20.8**	19.3
$\{700, 900, 1200, 1800\}$	25200	24.1	20.0	**21.3**	20.8	21.1	18.7
$\{700, 900, 1200\}$	25200	8.2	6.1	6.1	6.0	6.2	**7.4**
$\{720, 900, 1800, 3600\}$	3600	86.4	73.4	62.9	73.2	73.1	**83.7**
$\{720, 900, 3600\}$	3600	82.8	62.6	56.0	61.3	61.8	**81.1**
$\{800, 1600, 2400, 3200\}$	9600	80.3	74.9	72.2	**76.3**	75.3	60.6
$\{800, 1600, 2400\}$	9600	90.7	82.6	83.9	**84.7**	81.7	79.7
mean		47.3	40.2	38.6	40.4	40.2	**40.9**

Table 4.9 shows the results. Recall that the routing criterion functions of interest are the shortest paths (SP), load balancing (LB), and period matching (PM), as defined in Section 4.3.6. If the criterion function consists of two criteria, the primary criterion is written first and followed by the other, e.g., LBSP stands for LB as the primary criterion and SP as the secondary criterion. Although the Gurobi solver provides options for multicriteria optimization, this approach turned out to be slightly less efficient. Hence, we use the modeling approach where the primary criterion is multiplied by a sufficiently large constant (depending on the number of flows and links).

The results showed that PMSP performs best for the period sets with $HP = 3600$. Otherwise, we did not discover any direct relation between the period sets and the success of various routing criteria. In general, taking SP as the primary (or the only) criterion, which is mostly done in practice, seems to be a reasonable option to choose. The reason why taking other than the shortest paths is often detrimental can be explained by the fact that the TSN switches use the store-and-forward mechanism and that the switching delay is set to $10\mu s$.

We performed another set of experiments to show how the schedulability

Table 4.10: Percentage of scheduled problem instances if the cut-through mechanism is enabled

Periods	HP	Any	SP	LBSP	SPLB	SPPM	PMSP
$\{1400, 1800, 2400, 3600\}$	50400	26.2	22.1	23.3	23.1	22.4	**23.8**
$\{1400, 1800, 2400\}$	50400	10.1	8.3	8.1	8.1	8.6	**9.0**
$\{1400, 1800, 2800, 3600\}$	25200	41.0	33.6	33.9	**35.6**	34.0	29.0
$\{1400, 1800, 2800\}$	25200	27.9	23.3	22.8	23.0	**23.4**	18.8
$\{700, 900, 1200, 1800\}$	25200	23.9	**21.2**	**21.2**	21.0	21.0	20.8
$\{700, 900, 1200\}$	25200	9.0	7.4	7.3	7.3	**7.8**	**7.8**
$\{720, 900, 1800, 3600\}$	3600	91.6	73.7	63.2	71.2	72.4	**87.8**
$\{720, 900, 3600\}$	3600	87.8	64.7	57.7	61.8	61.7	**84.7**
$\{800, 1600, 2400, 3200\}$	9600	80.4	73.0	71.7	**74.7**	73.9	69.0
$\{800, 1600, 2400\}$	9600	96.3	87.8	88.8	**90.8**	88.4	86.2
mean		49.4	41.5	39.8	41.7	41.4	**43.7**

would increase if switches with the cut-through mechanism were used. In this case, we modify constraint 4.4 such that a transmission of a frame on a link does not need to wait until the frame on a preceding link is fully delivered to the switch and processed but can start right after 4 seconds since the preceding frame started its transmission, as assumed in [76]. The discussion on whether this assumption is realistic in TSN context is beyond the scope of this chapter. Table 4.10 confirms the expectation that the advantage of PMSP increased substantially more than for the other criteria since the cut-through mechanism enables paths with many hops to be scheduled. On the other hand, the overall increase in schedulability brought by the cut-through mechanism is not as significant as anticipated.

Finally, we ran the experiments without considering the frame isolation constraint in the model, i.e., without Constraint (4.5). Note that in this case, the CP1 and CP2 models are the same. The results showed that the mean schedulability that would be in column "Any" increased to 73.2 %. Moreover, for the period sets yielding HP of 3600 and 9600, the schedulability increased to 100 %. This confirms that the frame isolation constraint is the main burden in finding a feasible solution as the schedulability without this constraint would be significantly higher. We emphasize that the schedules crafted without this constraint are not valid deterministic schedules for TSN networks since the schedules neglect the fact that the transmissions of frames have to obey the FIFO property of the queues. However, these schedules may be used in the TTEthernet networks since the TTEthernet switches can dispatch the frames in arbitrary order.

4.6 Summary

This chapter addressed the problem of joint routing and scheduling in IEEE 802.1Qbv Time-Sensitive Networks. We developed two CP models: one with disjunctions and one with optional interval variables that represent the waiting of frames in queues. Further, we implemented approaches inspired by Logic-Based Benders Decomposition and pruning the search space by restricting the problem. The extensive experimental evaluation compared the various methods.

It has been shown that the simple CP model with disjunctions yields solutions of the best quality and is the best at proving the optimality of the found solutions. On the other hand, the CP model with interval variables representing waiting in queues to avoid disjunctions is very efficient in quickly finding a feasible solution. In particular, this model, combined with the Logic-Based Benders Decomposition, exhibited the best schedulability ratio. The overall success of the decomposition is a significant result showing that it allows for solving large-scale problems. Further, we investigated various criterion functions for the routing problem, thereby boosting the schedulability.

The results in this chapter were published in [6]. We also developed an efficient heuristic algorithm that solves the scheduling problems on large-scale networks (but disregards alternative routing options), which is published in [7]. Finally, we also investigated an option to enhance schedulability and throughput of the traffic in a network by enhancing the hardware of switches [8].

Conclusion

This thesis investigated the usage of optional activities. First, we coped with the scheduling problem of the production of water tubes where the machine reconfigurations lead to the sequence-dependent setups between consecutive tasks that must be performed by a machine setter who can only serve one machine at a time. Along with the efficient CP model using the optional activities to represent the setups between tasks, we presented four other CP models, an ILP formulation, and a hybrid heuristic that leverages the strength of ILP in the shortest Hamiltonian path problem and the efficiency of CP at sequencing problems with makespan minimization. The experimental evaluation showed that among the proposed exact approaches, the CP model with the optional activities is a superior method being able to solve instances with three machines and up to 11 tasks on each machine to optimality within a few seconds.

Another problem that we tackled is the MAPF problem, the goal of which is to find paths for agents. We approached MAPF using scheduling methodology where nodes and arcs are conceived as resources. We first described three models, where each agent can visit each node at most once. Then, we took the best model, which was the one that used optional activities to represent which nodes and arcs the agents will visit, and extended it so that agents could visit the nodes repetitively. The major contribution of the scheduling model of MAPF was its capability to accommodate other constraints. Namely, we extended the traditional MAPF formulation such that the capacities of arcs and the lengths of arcs could be greater than one. These extensions brought the model closer to real-life applications.

Finally, we addressed the problem of joint routing and scheduling of TT traffic in communication networks. Our approach computes routes of flows jointly with scheduling, which possibly allows more flows to be scheduled and hence increases the throughput in a network. We proposed two CP models with optional activities that represent the transmissions of data frames on communication links through which the data will be forwarded. An experimental evaluation comparing our solutions against the commonly used ILP and SMT formulations showed that our novel CP model, which also uses optional activities to represent waiting of frames in queues, brought a 50 % increase in the schedulability.

To conclude, this thesis focused on the notion of optional activities. It has been shown that the concept of optional activities can be easily applied not only to scheduling problems but also to various combinatorial optimization problems.

This thesis consists of the research results published during the Ph.D. studies of the author at Charles University.

Bibliography

[1] Marek Vlk, Antonin Novak, and Zdenek Hanzalek. Makespan minimization with sequence-dependent non-overlapping setups. In *International Conference on Operations Research and Enterprise Systems*, pages 91–101, 2019.

[2] Marek Vlk, Antonin Novak, Zdenek Hanzalek, and Arnaud Malapert. Non-overlapping sequence-dependent setup scheduling with dedicated tasks. In *International Conference on Operations Research and Enterprise Systems*, pages 23–46. Springer, 2019.

[3] Roman Barták, Jirı Švancara, and Marek Vlk. Scheduling models for multi-agent path finding. In *Proceedings of the Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA)*, pages 189–200, 2017.

[4] Roman Barták, Jiří Švancara, and Marek Vlk. A scheduling-based approach to multi-agent path finding with weighted and capacitated arcs. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems*, pages 748–756. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[5] Jiří Švancara, Marek Vlk, Roni Stern, Dor Atzmon, and Roman Barták. Online multi-agent pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7732–7739, 2019.

[6] Marek Vlk, Zdeněk Hanzálek, and Siyu Tang. Constraint programming approaches to joint routing and scheduling in time-sensitive networks. *Computers & Industrial Engineering*, 157, 2021.

[7] Marek Vlk, Kateřina Brejchová, Zdeněk Hanzálek, and Siyu Tang. Large-scale periodic scheduling in time-sensitive networks. *Computers & Operations Research*, 2021.

[8] Marek Vlk, Zdeněk Hanzálek, Kateřina Brejchová, Siyu Tang, Sushmit Bhattacharjee, and Songwei Fu. Enhancing schedulability and throughput of time-triggered traffic in IEEE 802.1 Qbv time-sensitive networks. *IEEE Transactions on Communications*, 68(11):7023–7038, 2020.

[9] Sally C Brailsford, Chris N Potts, and Barbara M Smith. Constraint satisfaction problems: Algorithms and applications. *European journal of operational research*, 119(3):557–581, 1999.

[10] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[11] Philippe Laborie, Jérôme Rogerie, Paul Shaw, and Petr Vilím. IBM ILOG CP optimizer for scheduling. *Constraints*, 23(2):210–250, 2018.

[12] Petr Vilím, Roman Barták, and Ondřej Čepek. Extension of o (n log n) filtering algorithms for the unary resource constraint to optional activities. *Constraints*, 10(4):403–425, 2005.

[13] Alexander Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, 1998.

[14] Gurobi. Optimizer reference manual. `http://www.gurobi.com/documentation/`.

[15] IBM ILOG Cplex. 12.2 User's Manual. *Book 12.2 User's Manual, Series 12.2 User's Manual*, 2010.

[16] Craig A Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984.

[17] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[18] Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535. ACM, 2001.

[19] João P Marques Silva and Karem A Sakallah. GRASP—a new search algorithm for satisfiability. In *Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*, pages 220–227. IEEE Computer Society, 1997.

[20] Roberto J Bayardo Jr and Robert Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *AAAI/IAAI: Proceedings of the fourteenth national conference on artificial intelligence and ninth conference on Innovative applications of artificial intelligence*, pages 203–208, 1997.

[21] Justyna Petke. *Bridging Constraint Satisfaction and Boolean Satisfiability.* Springer, 2015.

[22] Olga Ohrimenko, Peter J Stuckey, and Michael Codish. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, 2009.

[23] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL (T). *Journal of the ACM (JACM)*, 53(6):937–977, 2006.

[24] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[25] Bruno Dutertre and Leonardo De Moura. The yices SMT solver. *Tool paper at https://yices.csl.sri.com/papers/tool-paper.pdf*, 2(2):1–2, 2006.

[26] Philippe Laborie, Jerome Rogerie, Paul Shaw, and Petr Vilím. Reasoning with conditional time-intervals. part ii: An algebraical model for resources. In *FLAIRS conference*, pages 201–206, 2009.

[27] Ali Allahverdi, CT Ng, TC Edwin Cheng, and Mikhail Y Kovalyov. A survey of scheduling problems with setup times or costs. *European journal of operational research*, 187(3):985–1032, 2008.

[28] Young Hoon Lee and Michael Pinedo. Scheduling jobs on parallel machines with sequence-dependent setup times. *European Journal of Operational Research*, 100(3):464–474, 1997.

[29] Eva Vallada and Rubén Ruiz. A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, 211(3):612–622, 2011.

[30] Rubén Ruiz and Carlos Andrés-Romano. Scheduling unrelated parallel machines with resource-assignable sequence-dependent setup times. *The International Journal of Advanced Manufacturing Technology*, 57(5-8):777–794, 2011.

[31] Horst Tempelmeier and Lisbeth Buschkühl. Dynamic multi-machine lotsizing and sequencing with simultaneous scheduling of a common setup resource. *International Journal of Production Economics*, 113(1):401–412, 2008.

[32] Dong Chen, Peter B Luh, Lakshman S Thakur, and Jack Moreno Jr. Optimization-based manufacturing scheduling with multiple resources, setup requirements, and transfer lots. *IIE Transactions*, 35(10):973–985, 2003.

[33] Xing Zhao, Peter B Luh, and Jihua Wang. Surrogate gradient algorithm for lagrangian relaxation. *Journal of optimization Theory and Applications*, 100(3):699–712, 1999.

[34] Erick D. Wikum, Donna C. Llewellyn, and George L. Nemhauser. One-machine generalized precedence constrained scheduling problems. *Operations Research Letters*, 16(2):87 – 99, 1994.

[35] Jean B Lasserre and Maurice Queyranne. Generic scheduling polyhedra and a new mixed-integer formulation for single-machine scheduling. *Proceedings of the 2nd IPCO (Integer Programming and Combinatorial Optimization) conference*, pages 136–149, 1992.

[36] David Applegate and William Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on computing*, 3(2):149–156, 1991.

[37] Egon Balas. Project scheduling with resource constraints. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1968.

[38] David L Applegate, Robert E Bixby, Vasek Chvátal, and William J Cook. *The Traveling Salesman Problem: A Computational Study.* Princeton University Press, 2011.

[39] Ulrich Pferschy and Rostislav Staněk. Generating subtour elimination constraints for the TSP from pure integer solutions. *Central European Journal of Operations Research*, 25(1):231–260, Mar 2017.

[40] Pascal Van Hentenryck and Laurent Michel. *Constraint-based local search.* The MIT press, 2009.

[41] David Pisinger and Stefan Ropke. Large neighborhood search. In *Handbook of metaheuristics*, pages 399–419. Springer, 2010.

[42] Google. Or-tools. `https://developers.google.com/optimization/`.

[43] Malcolm Ross Kinsella Ryan. Exploiting subgraph structure in multi-robot path planning. *Journal of Artificial Intelligence Research*, 31:497–542, 2008.

[44] Daniel Martin Kornhauser, Gary Miller, and Paul Spirakis. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. Master's thesis, M. I. T., Dept. of Electrical Engineering and Computer Science, 1984.

[45] Dong-Gyun Kim, Katsutoshi Hirayama, and Gyei-Kark Park. Collision avoidance in multiple-ship situations by distributed local search. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 18(5):839–848, 2014.

[46] Nathan Michael, Jonathan Fink, and Vijay Kumar. Cooperative manipulation and transportation with aerial robots. *Autonomous Robots*, 30(1):73–86, 2011.

[47] Jur van Den Berg, Jack Snoeyink, Ming C Lin, and Dinesh Manocha. Centralized path planning for multiple robots: Optimal decoupling into sequential plans. In *Robotics: Science and systems*, volume 2, pages 2–3, 2009.

[48] Ko-Hsin Cindy Wang, Adi Botea, et al. Fast and memory-efficient multi-agent pathfinding. In *ICAPS*, pages 380–387, 2008.

[49] Daniel Ratner and Manfred Warmuth. The (n2- 1)-puzzle and related relocation problems. *Journal of Symbolic Computation*, 10(2):111–137, 1990.

[50] Pavel Surynek. Towards optimal cooperative path planning in hard setups through satisfiability solving. In *Pacific Rim International Conference on Artificial Intelligence*, pages 564–576. Springer, 2012.

[51] Jingjin Yu and Steven M LaValle. Planning optimal paths for multiple robots on graphs. In *2013 IEEE International Conference on Robotics and Automation*, pages 3612–3617. IEEE, 2013.

[52] Esra Erdem, Doga Gizem Kisa, Umut Oztok, and Peter Schüller. A general formal framework for pathfinding problems with multiple agents. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[53] Trevor Standley. Finding optimal solutions to cooperative pathfinding problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, 2010.

[54] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

[55] Eli Boyarski, Ariel Felner, Roni Stern, Guni Sharon, David Tolpin, Oded Betzalel, and Eyal Shimony. ICBS: Improved conflict-based search algorithm for multi-agent pathfinding. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[56] Guni Sharon, Roni Stern, Meir Goldenberg, and Ariel Felner. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence*, 195:470–495, 2013.

[57] Roman Barták, Agostino Dovier, and Neng-Fa Zhou. Multiple-origin-multiple-destination path finding with minimal arc usage: complexity and models. In *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 91–97. IEEE, 2016.

[58] Ryan J Luna and Kostas E Bekris. Push and swap: Fast cooperative path-finding with completeness guarantees. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[59] Roman Barták, Neng-Fa Zhou, Roni Stern, Eli Boyarski, and Pavel Surynek. Modeling and solving the multi-agent pathfinding problem in picat. In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 959–966. IEEE, 2017.

[60] Robert W Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.

[61] Pavel Surynek. A novel approach to path planning for multiple robots in bi-connected graphs. In *2009 IEEE International Conference on Robotics and Automation*, pages 3613–3619. IEEE, 2009.

[62] Silviu S Craciunas, Ramon Serna Oliver, Martin Chmelík, and Wilfried Steiner. Scheduling real-time communication in IEEE 802.1 Qbv time sensitive networks. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, pages 183–192. ACM, 2016.

[63] Eike Schweissguth, Peter Danielis, Dirk Timmermann, Helge Parzyjegla, and Gero Mühl. ILP-based joint routing and scheduling for time-triggered networks. In *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, pages 8–17. ACM, 2017.

[64] Industrial Internet Consortium et al. Time sensitive networks for flexible manufacturing testbed: Characterization and mapping of converged traffic types. Technical report, 2019.

[65] Chun-Cheng Lin, Lun-Ping Hung, Wan-Yu Liu, and Ming-Chun Tsai. Jointly rostering, routing, and rerostering for home health care services: A harmony search approach with genetic, saturation, inheritance, and immigrant schemes. *Computers & Industrial Engineering*, 115:151–166, 2018.

[66] Asefeh Hasani Goodarzi, Reza Tavakkoli-Moghaddam, and Alireza Amini. A new bi-objective vehicle routing-scheduling problem with cross-docking: mathematical model and algorithms. *Computers & Industrial Engineering*, 149:106832, 2020.

[67] Denise Yamashita, Bruno Jensen Virginio da Silva, Reinaldo Morabito, and Paulo Cesar Ribas. A multi-start heuristic for the ship routing and scheduling of an oil company. *Computers & Industrial Engineering*, 136:464–476, 2019.

[68] Mohsen Babaei and Mojtaba Rajabi-Bahaabadi. School bus routing and scheduling with stochastic time-dependent travel times considering on-time arrival reliability. *Computers & Industrial Engineering*, 138:106125, 2019.

[69] Benjamin Caddell. Joint routing and scheduling with SMT. B.S. thesis, University of Stuttgart, 2018.

[70] Fedor Smirnov, Michael Glaß, Felix Reimann, and Jürgen Teich. Optimizing message routing and scheduling in automotive mixed-criticality time-triggered networks. In *Proceedings of the 54th Annual Design Automation Conference 2017*, page 48. ACM, 2017.

[71] Jonathan Falk, Frank Dürr, and Kurt Rothermel. Exploring practical limitations of joint routing and scheduling for TSN with ILP. In *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 136–146. IEEE, 2018.

[72] Naresh Ganesh Nayak, Frank Dürr, and Kurt Rothermel. Routing algorithms for IEEE802. 1Qbv networks. *ACM SIGBED Review*, 15(3):13–18, 2018.

[73] Frank Dürr and Naresh Ganesh Nayak. No-wait packet scheduling for IEEE time-sensitive networks (TSN). In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, pages 203–212. ACM, 2016.

[74] Ayman Atallah, Ghaith Bany Hamad, and Otmane Ait Mohamed. Routing and scheduling of time-triggered traffic in time sensitive networks. *IEEE Transactions on Industrial Informatics*, 2019.

[75] Qinghan Yu and Ming Gu. Adaptive group routing and scheduling in multicast time-sensitive networks. *IEEE Access*, 8:37855–37865, 2020.

[76] Eike Schweissguth, Dirk Timmermann, Helge Parzyjegla, Peter Danielis, and Gero Mühl. ILP-based routing and scheduling of multicast realtime traffic in time-sensitive networks. In *2020 IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 1–11. IEEE, 2020.

[77] Paul Pop, Michael Lander Raagaard, Silviu S Craciunas, and Wilfried Steiner. Design optimisation of cyber-physical distributed systems using IEEE time-sensitive networks. *IET Cyber-Physical Systems: Theory & Applications*, 1(1):86–94, 2016.

[78] Sune Mølgaard Laursen, Paul Pop, and Wilfried Steiner. Routing optimization of AVB streams in TSN networks. *ACM Sigbed Review*, 13(4):43–48, 2016.

[79] Thomas A Feo and Mauricio GC Resende. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133, 1995.

[80] Voica Gavriluţ, Luxi Zhao, Michael L Raagaard, and Paul Pop. AVB-aware routing and scheduling of time-triggered traffic for TSN. *IEEE Access*, 6:75229–75243, 2018.

[81] Jin Y Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971.

[82] Mubarak Adetunji Ojewale and Patrick Meumeu Yomsi. Routing heuristics for load-balanced transmission in TSN-based networks. *ACM Sigbed Review*, 16(4):20–25, 2020.

[83] Naresh Ganesh Nayak, Frank Dürr, and Kurt Rothermel. Time-sensitive software-defined network (TSSDN) for real-time applications. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, pages 193–202. ACM, 2016.

[84] Naresh Ganesh Nayak, Frank Dürr, and Kurt Rothermel. Incremental flow scheduling and routing in time-sensitive software-defined networks. *IEEE Transactions on Industrial Informatics*, 14(5):2066–2075, 2018.

[85] Jonas Diemer, Daniel Thiele, and Rolf Ernst. Formal worst-case timing analysis of Ethernet topologies with strict-priority and AVB switching. In *7th IEEE International Symposium on Industrial Embedded Systems (SIES'12)*, pages 1–10. IEEE, 2012.

[86] Luxi Zhao, Paul Pop, and Silviu S Craciunas. Worst-case latency analysis for IEEE 802.1 Qbv time sensitive networks using network calculus. *IEEE Access*, 6:41803–41815, 2018.

[87] Henri Bauer, Jean-Luc Scharbarg, and Christian Fraboul. Improving the worst-case delay analysis of an afdx network using an optimized trajectory approach. *IEEE Transactions on Industrial informatics*, 6(4):521–533, 2010.

[88] Karl-Erik Årzén and Anton Cervin. Control and embedded computing: Survey of research directions. *IFAC Proceedings Volumes*, 38(1):191–202, 2005.

[89] Ramon Faganello Fachini and Vinícius Amaral Armentano. Logic-based benders decomposition for the heterogeneous fixed fleet vehicle routing problem with time windows. *Computers & Industrial Engineering*, 148:106641, 2020.

[90] Jheng-Yu Huang, Ming-Hung Hsu, and Chung-An Shen. A novel routing algorithm for the acceleration of flow scheduling in time-sensitive networks. *Sensors*, 20(21):6400, 2020.

[91] Giorgio C Buttazzo. Rate monotonic vs. EDF: judgment day. *Real-Time Systems*, 29(1):5–26, 2005.

List of Abbreviations

AVB Audio Video Bridging

COP constraint optimization problem

CP Constraint Programming

CSP constraint satisfaction problem

FIFO first-in-first-out

GCL gate control list

GRASP Greedy Randomized Adaptive Search Procedure

ILP Integer Linear Programming

JRAS joint routing and scheduling

LOFAS Local Optimization for Avoided Setup

LP Linear Programming

MAPF multi-agent path finding

MILP Mixed Integer Linear Programming

MIP Mixed Integer Programming

OMT Optimization Modulo Theories

SAT Boolean Satisfiability

SMT Satisfiability Modulo Theories

SDN software-defined networking

TAS time-aware shaper

TSN Time-Sensitive Networking

TSSDN time-sensitive software-defined networks

TT time-triggered

List of Publications

2021

- Vlk M. (45 %), Brejchová K., Hanzálek Z., Tang S. Large-Scale Periodic Scheduling in Time-Sensitive Networks. In *Computers & Operations Research*, 2021.
- Vlk M. (90 %), Hanzálek Z., Tang S. Constraint Programming Approaches to Joint Routing and Scheduling in Time-Sensitive Networks. In *Computers & Industrial Engineering*, 2021.

2020

- Vlk M. (50 %), Hanzálek Z., Brejchová K., Tang S., Bhattacharjee S., Fu S. Enhancing Schedulability and Throughput in IEEE 802.1Qbv Time-Sensitive Networks. In *IEEE Transactions on Communications*, pages 7023–7038. 2020.
- Vlk M. (50 %), Novák A., Hanzálek Z., Malapert A. Non-overlapping Sequence-Dependent Setup Scheduling with Dedicated Tasks. In *Parlier G., Liberatore F., Demange M. (eds) Operations Research and Enterprise Systems. ICORES 2019. Communications in Computer and Information Science*, pages 23–46, 2020.

2019

- Vlk M. (60 %), Novák A., Hanzálek Z. Makespan Minimization with Sequence-Dependent Non-overlapping Setups. In *ICORES 2019 - Proceedings of the International Conference on Operations Research and Enterprise Systems*, pages 91–101, 2019.
- Švancara J., Vlk M. (20 %), Stern R., Atzmon D., Barták R. Online Multi-agent Pathfinding. In *AAAI 2019 - Proceedings of the AAAI Conference on Artificial Intelligence*, pages 7732–7739, 2019.

2018

- Barták R., Švancara J., Vlk M. (70 %) A Scheduling-Based Approach to Multi-agent Path Finding with Weighted and Capacitated Arcs. In *AAMAS 2018 - Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 748–756, 2018.
- Benedikt O., Šůcha P., Módos I., Vlk M. (10 %), Hanzálek Z. Energy-Aware Production Scheduling with Power-Saving Modes. In *CPAIOR 2018 - Proceedings of the International Conference on the Integration of Constraint*

Programming, Artificial Intelligence, and Operations Research, pages 72–81, 2018.

2017

- Barták R., Švancara J., Vlk M. (50 %) Scheduling Models for Multi-Agent Path Finding. In *MISTA 2017 - Proceedings of the Multidisciplinary International Conference on Scheduling : Theory and Applications*, pages 189–200, 2017.
- Vlk M. (90 %), Barták R., Hebrard E. Benders Decomposition in SMT for Rescheduling of Hierarchical Workflows. In *MISTA 2017 - Proceedings of the Multidisciplinary International Conference on Scheduling : Theory and Applications*, 2017.
- Vlk M. (90 %), Barták R., Hanzálek Z. Minimization of Useless Work in Resource Failure Recovery of Workflow Schedules. In *ETFA 2017 - Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation*, 2017.

2016

- Barták R., Vlk M. (95 %) Hierarchical Task Model for Resource Failure Recovery in Production Scheduling. In Proceedings of MICAI 2016, *Lecture Notes in Artificial Intelligence (LNAI) Series*, pages 362–378, 2016.

2015

- Barták R., Vlk M. (95 %) Machine Breakdown Recovery in Production Scheduling with Simple Temporal Constraints. In *Lecture Notes in Artificial Intelligence (LNAI) Series*, pages 185–206, 2015.
- Barták R., Vlk M. (95 %) Resource Failure Recovery in Production Scheduling. In *MISTA 2015 - Proceedings of the Multidisciplinary International Conference on Scheduling : Theory and Applications*, pages 852–858, 2015.
- Barták R., Vlk M. (95 %) Reactive Recovery from Machine Breakdown in Production Scheduling with Temporal Distance and Resource Constraints. In *ICAART 2015 - Proceedings of the International Conference on Agents and Artificial Intelligence*, pages 119–130, 2015.