THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Modular Learning and Optimization for Planning of Discrete Event Systems

Fredrik Hagebring

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2021

Modular Learning and Optimization for Planning of Discrete Event Systems

Fredrik Hagebring
ISBN 978-91-7905-555-4

Doktorsavhandlingar vid Chalmers tekniska högskola
Ny serie nr 5022
ISSN 0346-718X

Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg, Sweden
Phone: +46 (0)31 772 1000

Printed by Chalmers digitaltryck
Gothenburg, Sweden, October 2021

Modular Learning and Optimization for Planning of Discrete Event Systems

Fredrik Hagebring
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Optimization of industrial processes, such as manufacturing cells, can have great impact on their performance. Finding optimal solutions to these large-scale systems is, however, a complex problem. They typically include multiple subsystems, and the search space generally grows exponentially with each subsystem. This is usually referred to as the *state explosion problem* and is a well-known problem within the control and optimization of automation systems.

This thesis proposes two main contributions to improve and to simplify the optimization of these systems. The first is a new method of solving these optimization problems using a compositional optimization approach. This integrates optimization with techniques from compositional supervisory control using modular formal models, dividing the optimization of subsystems into separate subproblems. The second is a modular learning approach that alleviates the need for prior knowledge of the systems and system experts when applying compositional optimization.

The key to both techniques is the division of the large system into smaller subsystems and the identification of local behavior in these subsystems, i.e. behavior that is independent of all other subsystems. It is proven in this thesis that this local behavior can be partially optimized individually without affecting the global optimal solution. This is used to reduce the state space in each subsystem, and to construct the global optimal solution compositionally.

The thesis also shows that the proposed techniques can be integrated to compute global optimal solutions to large-scale optimization problems, too big to solve based on traditional monolithic models.

Keywords: Compositional optimization, large-scale optimization, automation, modular learning, active learning, discrete event systems, discrete optimization.

List of Publications

This thesis is based on the following publications:

[A] **Fredrik Hagebring**, Bengt Lennartson, "Compositional optimization of discrete event systems". Proceedings of the *14th IEEE Conference on Automation Science and Engineering*, 2018.

[B] **Fredrik Hagebring**, Bengt Lennartson, "Time-optimal control of large-scale systems of systems using compositional optimization". *Discrete Event Dynamic Systems: Theory and Applications*, Sep. 2019.

[C] Ashfaq Farooqui, **Fredrik Hagebring**, Martin Fabian, "Active learning of modular plant models". Proceedings of the *15th IFAC Workshop on Discrete Event Systems*, 2020.

[D] **Fredrik Hagebring**, Ashfaq Farooqui, Martin Fabian, "Modular supervisory synthesis for unknown plant models using active learning". Proceedings of the *15th IFAC Workshop on Discrete Event Systems*, 2020.

[E] **Fredrik Hagebring**, Ashfaq Farooqui, Martin Fabian, Bengt Lennartson, "On optimization of automation systems: integrating modular learning and optimization". Submitted for possible publication in *IEEE Transactions on Automation Science and Engineering*, 2021.

Other publications by the author, not included in this thesis, are:

[F] **Fredrik Hagebring**, Oskar Wigström, Bengt Lennartson, Simon Ian Ware, Rong Su, "Comparing MILP, CP, and A* for multiple stacker crane scheduling". Proceedings of the *13th International Workshop on Discrete Event Systems*, 2016, Xi'an, China.

[G] Sabino Francesco Roselli, **Fredrik Hagebring**, Sarmad Riazi, Martin Fabian, Knut Åkesson, "On the use of equivalence classes for optimal and sub-optimal bin covering". Proceedings of the *15th IEEE International Conference on Automation Science and Engineering*, 2019, Vancouver, Canada.

[H] Sabino Francesco Roselli, **Fredrik Hagebring**, Sarmad Riazi, Martin Fabian, Knut Åkesson, "On the use of equivalence classes for optimal and

suboptimal bin packing and bin covering". *IEEE Transactions on Automation Science and Engineering*, 2020, Early access.
DOI: 10.1109/TASE.2020.3022986.

[I] Ashfaq Farooqui, **Fredrik Hagebring**, Martin Fabian, "MIDES: A Tool for Supervisor Synthesis via Active Learning". Accepted to *IEEE CASE*, 2021.

Acronyms

MILP:	Mixed Integer Linear Programming
LP:	Linear Programming
CP:	Constraint Programming
SCT:	Supervisory Control Theory
CompOpt:	Compositional Optimization
DES:	Discrete Event System
DFA:	Deterministic Finite Automata
NFA:	Non-deterministic Finite Automata
PTWS:	Partial Time-Weighted Synchronization
PSH:	Plant Structure Hypothesis
MPL:	Modular Plant Learner
MSL:	Modular Supervisor Learner
MOL:	Modular Optimization Learner
MAL:	Modular Active Learning
MDP:	Markov Decision Process

Contents

Part I

Overview

CHAPTER 1

Introduction

Automation is the technology by which a process or procedure is performed with minimum human assistance [1]. This is something that we find everywhere in today's society, within production, logistics, communication and countless other areas. Moreover, the level of automation in these areas is continuously increasing with new, more advanced tasks being automated in a pursuit of an increased performance of the system. However, with an increased level of automation comes an increased complexity. To design a controller for a single automated task can be tricky enough, and to derive a plan that coordinates hundreds of tasks is not a trivial problem.

The planning of an automation system can, for simplicity, be described in two parts: (i) to ensure that all tasks are performed correctly and that the goal is finally achieved, and (ii) to maximize the performance of the complete process such that it can be executed as fast or as cheap as possible. The second part is an extension of the first, and implies that it is no longer enough to reach the goal state. Instead, the plan needs to be optimized to ensure that the process can be executed in the best way possible.

A wide range of efficient methods for solving this type of optimization problems have been explored over the years. There are a wide range of different

optimization techniques. For further references we recommend Passino and Antsaklis [2], Brandin and Wonham [3], Huang and Kumar [4], Kobetski and Fabian [5] and Hagebring *et al.* [6]. Many of these techniques have been proven efficient with respect to computational complexity and typically scale polynomially with the size of the system [2]–[6]. Moreover, the type of problems that are addressed can typically be perceived to relate directly to other large fields of optimization research, such as planning and scheduling. For example, *markov decision process* theory (MDP), which is the most basic modeling tool for stochastic scheduling, is claimed in the textbook by Cassandras and Lafortune [7] as a formal DES framework. Regardless of the modelling tool or the solution method, all methods suffer from the well-known *state explosion problem*, also called the *curse of dimensionality* [8], [9]. The problem of addressing this task has of course been investigated in a large number of publications. For further references we recommend Powell [10], Cao [11], Bertsekas and Tsitsiklis [12] and Bertsekas [13].

The work presented in this thesis introduces a novel optimization technique for large-scale automation systems. It is specifically designed to be used for planning of systems that are separable into subsystems (systems of systems), and exploits this property to divide the optimization into multiple independent subproblems. The thesis also presents work on a learning technique that allows the planning to be conducted without the existence of any predefined formal models. The work focuses mainly on production and logistics systems, where multiple robots collaborate to perform certain tasks. The same theories could, however, be applied to any automation system that is modeled similarly.

1.1 Background

The journey began in 2015 when I was asked to help with the optimization of a logistics system, including three stacker cranes that should collaborate to perform a number of tasks [6]. Though the problem was simplified, the complexity of the optimization problem was proved to be hard to solve efficiently. This problem is not caused mainly by the formulation of the optimization model or the choice of solver. Instead, the problem with the optimization of automation systems is that the size of their state space suffers from the state explosion problem. Consequently, it is not enough with methods that scales polynomially with the size of the system if the size of the system scales

exponentially with the number and size of its subsystems.

To mitigate this problem I started to look for viable methods do decrease the complexity of the system even before the optimization took part. A background in supervisory control theory (SCT) lead me to investigate if efficient methods from verification and synthesis could be used to prune all infeasible solutions so that they do not have to be considered by the optimization. While this alone could probably reduce the complexity enough to solve slightly larger systems, it also raised the question if the modular or compositional algorithms from SCT [14], [15] could be applied also in the optimization. These algorithms operate on a *modular model* in which the model of the complete system has been divided into multiple smaller models, *modules*, that represent different subsystems in the system. The synchronous composition [7] of all the modules results in the complete behaviour of the system. Compositional supervisory control can efficiently verify and synthesize controllers for large-scale systems [16], [17]. In recent years a few modular or compositional approaches to optimization have also been presented, but these are either very restrictive in their reduction of the subsystems or offer only approximative solutions [18]–[20]. This became the inspiration to my research project *Integrated Verification, Optimization and Learning*, in which I wanted to further evaluate compositional approaches to optimization of large-scale automation systems.

1.2 Research questions

The strength of a compositional approach comes from the ability to reduce each module individually while synchronizing them compositionally into one final system model. The goal is to remove as many states and transitions as possible in each iteration without altering the final solution. Compositional supervisory synthesis constructs maximally permissive controllers for a given set of specifications [7]. These controllers should ensure that something bad never happens, while leaving everything else unchanged. In this case, the individual modules are reduced by removing all “bad states”, in which the specifications are not satisfied. It is easy to see how this reduction leaves the final solution unchanged, since the final supervisor must prevent something bad from happening in all included subsystems. Thus, since a “bad state” in an individual module represents something bad happening in the correspond-

ing subsystem, it will either be removed automatically in the synchronization of the modules or will have to be removed in a later stage of the synthesis. In optimization, this synthesis is equivalent to the removal of all infeasible solutions. This can of course be useful, but applying a compositional approach to optimization implies that the reduction of the modules also removes suboptimal behavior, such that the synchronous composition corresponds to the globally optimal solution. Proving what parts of an individual module that are suboptimal is, however, not trivial, which leads to the first research question:

1. *Can optimization of system of systems be done compositionally and still guarantee a global optimal solution?*

This question became the inspiration to *Compositional Optimization* (CompOpt) presented in Papers A and B, which uses modular automata models [14] to identify which parts that can be optimized locally. While this technique is considered a major contribution of the thesis, its applicability in industry is limited by its reliance on existing formal models of the system. Many industries have yet to adopt model-based techniques, thus often lack both the expertise and the system knowledge required to formulate the modular models needed for the compositional approaches in SCT and CompOpt. This motivated the the second research question:

2. *Can a modular model be learnt automatically for a system of systems?*

Answering this question led to the discovery of a *modular active learning* (MAL) technique. This was first demonstrated by the two methods: *Modular Plant Learner* (MPL) in Paper C that learns modular plant models, and *Modular Supervisor Learner* (MSL) in Paper D that learns modular supervisors for the system using a set of predefined specifications (MSL). My personal interest in the development of MPL was to relieve CompOpt of the reliance on existing plant models, and instead letting the necessary models be learnt automatically. This work did, however, raise the question of whether an integrated MAL and CompOpt approach could further improve the efficiency of the optimization, compared to a sequential computation. That is:

3. *Can a combination of MAL and CompOpt yield a synergy that allows for efficient optimal planning of large-scale automation systems without the existence of formal models?*

This led to the most recent and final contribution presented in this thesis – the *Modular Optimization Learner* (MOL) in Paper E – that learns modular optimization models to be used in CompOpt.

1.3 Contributions

The main contributions presented in this thesis are:

1. The *Compositional Optimization* framework (CompOpt) that specializes in the optimization of large-scale systems of systems. Similarly to compositional synthesis in SCT, CompOpt reduces each module individually using a *local optimization* technique, while synchronizing them compositionally into the global optimal solution. Its strength comes from the ability to identify suboptimal paths in the individual subsystems. These can then be removed to reduce the state spaces of the modules, and consequently, mitigating the state explosion of their synchronous composition. Results in Papers A and B show that CompOpt drastically reduces the search space during the optimization of a realistic large-scale example and, hence, improves the computational complexity.

2. The *Modular Active Learning* technique (MAL) that learns modular formal models of systems of systems. Similarly to CompOpt, MAL is inspired by techniques in SCT to divide large systems into modules that represent individual subsystems. The strength of a modular representation is that: (i) the learning algorithm in many cases can learn a complete system model while only exploring fractions of the state space and (ii) it allows for a compact modelling of large-scale systems that otherwise would include millions of states. It is shown in Papers C, D and E that MAL can be customized to learn models with specific properties such as maximally permissive plant models, controllable supervisors, and modular optimization models, respectively.

3. The Modular Optimization Learner (MOL), specifically designed to learn efficient modular optimization models for CompOpt. This integrates MAL with a greedy search heuristic that identifies suboptimal solutions, which can then be removed from the final modular optimization model. The result in Paper E shows that this drastically reduces the

state space of the models and, consequently, the search space of the subsequent optimization. This completes the CompOpt framework, allowing for optimal scheduling of large-scale systems of systems, even with minimal prior system knowledge.

1.4 Outline

This thesis is divided into two parts. The first part aims to give the reader an overview of the field of research and a better understanding of the concepts discussed in the included papers. The included papers constitute the second part.

The introduction provides the background and research questions that have been the inspiration for the work. Chapter 2 focuses on the concept of modular/compositional optimization. Chapter 3 introduces the modular active learning in MAL and discusses the properties of a modular system model. Chapter 4 presents the integration of MAL and CompOpt into a joint framework. A summary of the included papers is provided in Chapter 5. Finally, concluding remarks and a direction for future work are provided in Chapter 6.

CHAPTER 2

Compositional optimization

The aim of this chapter is to explain the concept of compositional optimization in general, the challenges and the potential it presents, and finally to give the reader a better understanding of CompOpt (the compositional optimization method presented in this thesis) through a set of examples that illustrate the key properties of the technique.

The main goal of compositional optimization is to minimize the state explosion problem for the optimization of systems consisting of multiple subsystems. The basic concept is to reduce each subsystem individually by exploiting *local* behaviour and then construct the global solution compositionally using their reduced models. This enables a global optimization of the complete automation system without considering the complete monolithic state space. The benefit is that this has the potential to reduce the overall complexity, since the computation of a monolithic model typically results in a state explosion, where the state space scales exponentially with the size and number of subsystems.

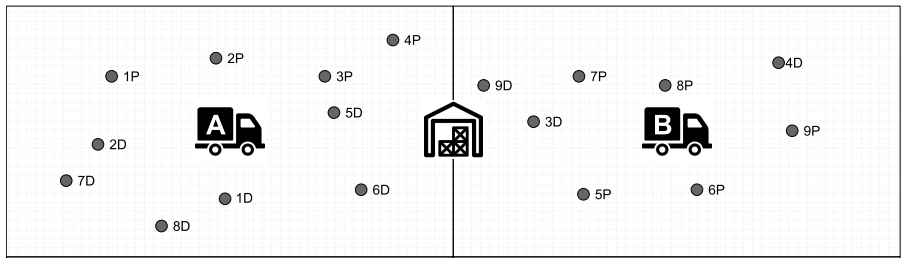


Figure 2.1: Illustration of a simple logistics system consisting of two delivery trucks, A and B, operating in adjacent neighbourhoods, that should pick-up and deliver a total of nine packages. The pick up and delivery location of a package i is marked iP and iD respectively.

2.1 Motivating example

This section provides a motivating example to illustrate the impact of the state explosion problem and the potential benefits and challenges of using a compositional optimization approach. The example depicts a simple logistics system, consisting of two delivery trucks that pick up and deliver packages in separate zones. Every day, there are a list of packages that should be picked up and delivered within there operation area. One could argue that this motivating example does not really depict the optimization of a large-scale system of systems, but the example is in fact already large enough for the purpose of this illustration. The system is illustrated in Fig. 2.1.

The figure shows the two trucks and their respective zones. In the center of the area there is a warehouse, which is where the trucks must start and end each day. The figure also includes an example of a scenario where nine packages should be picked up and delivered during the day. The pick up and delivery location of these packages are marked with dots on the map, where the labels iP and iD represent the pick up and delivery locations of package i, respectively. Some packages should be picked up in one zone but delivered in another. In these cases the truck that picks up the package has to bring it back to the central warehouse where it can be moved over to the delivery truck. These types of switches between the trucks are assumed to occur only once a day. The objective in this example is to deliver all packages as quickly as possible, that is, the goal is to minimize the time it will take the last truck to return to the warehouse in the afternoon. The weights to be considered by

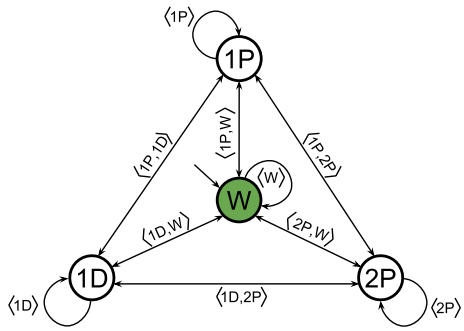
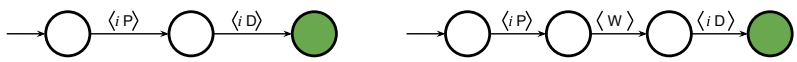


Figure 2.2: An automata model of the possible behavior of truck A, when assigned the tasks to pick up packages 1, 2 and deliver package 1. States represent the physical locations, while edges represent operations in these locations and travel in between. The state W represents the central warehouse, which is both the initial and the accepting state of the model.

the cost function should in this case represent the time it takes to perform each task. The tasks include the pick up and delivery of packages, as well as the travel between these locations.

The physical position of each truck, can be modelled as a strongly connected graph, where nodes represent the locations of the warehouse and the pick up/delivery tasks, while the edges represent the travel in between. The actual pick up and delivery operations can be modelled as self loops in the nodes of the graph, indicating that a task is performed but the physical location does not change. In favor of readability, a reduced example where truck A only has to pick up and deliver package 1 and pick up package 2 is modelled using a simple automaton in Fig.2.2. The markings of the transitions are: (i) the self loops marked by $\langle x \rangle$ illustrating the different operations that can be performed in each location, including $\langle W \rangle$, which represents that the trucks switch packages at the central warehouse, and (ii) the edges between different locations marked by $\langle x, y \rangle$ representing the travel between two locations x and y. The central warehouse is marked green to illustrate that this is the desired goal state, the *accepting* state.

In addition to a model of the *possible* behavior, there are of course also models of the *desired* behavior. These are specified in Fig. 2.3. The specifi-



(a) **(b)**

Figure 2.3: Generalized models of individual specifications for the route of each package. (a) applies to packages that are picked up and delivered by the same truck, (b) applies to packages that should be picked up by one truck and delivered by another.

cation in Fig. 2.3(a) is applied to all packages that should be picked up and delivered by the same truck. It specifies that the package has to be picked up and then delivered to its final delivery location exactly once. The specification in Fig. 2.3(b) is similar to 2.3(a) but should be applied whenever a package is to be picked up in one zone by truck X and delivered to another zone by truck Y. It is then required that the package is switched from one truck to the other in the central warehouse. Individual specifications like these have to be included for each package.

To evaluate the example, the scenario from Fig 2.1 is modelled as a system of systems, using plant models for each truck and specifications for each package to represent the subsystems, such as shown in Fig. 2.2 and Fig. 2.3. Any optimization applied using a monolithic approach would have to consider a search space spanning the complete synchronized behavior of all subsystems. This is true regardless of the optimization paradigm that is used. Advanced paradigms, such as MILP, CP, might be able to perform clever pruning of the search space in an early stage, but initially all possible combinations of states and transitions have to be considered. This is a potential problem since the size of the search space grows exponentially due to the state explosion problem. The search space of the simple example shown here includes 342 144 states and 6 329 115 transitions, representing the synchronous composition of all subsystems.

When solving the same example using CompOpt, the optimization problem is partitioned into multiple subproblems but the sum of states in the search spaces of all subproblems combined only adds up to 16 396 states. The reason that CompOpt is able to perform so much better than the monolithic approach

is the ability to reduce the subsystems even before they are synchronized. The full search space is never computed, therefore no unnecessary states have to be pruned away or evaluated. It is worth noting that CompOpt only represents one specific compositional approach, which most certainly can be further enhanced, but the purpose of this example is just to illustrate that there is much to gain from the ability to optimize systems of systems compositionally.

One could argue that there might exist more efficient models of this system than what is shown here. To a human it is for example obvious that the trucks can be partially optimized individually, since they drive in separate areas, have separate lists of tasks and so on. It is, however, not obvious exactly how this problem can be partitioned since there still exist dependencies between the trucks. Without digging into the details of exactly which tasks that can be considered local, there is no way to partition this problem manually. One benefit of using CompOpt is that it reduces the need of *smart* manual partitioning of the optimization problem, since it already exploits the local behavior maximally.

2.2 Challenges

Discrete optimization problems are considered NP-hard or NP-complete [21] in general. Therefore, there cannot exist any general method to solve all of these in polynomial time. There is simply no way to completely solve the state explosion problem. To solve large-scale optimization problems one must instead exploit the properties of the problem. Efficient solution methods have been presented over the years for several classes of discrete optimization problems, such as traveling salesman, graph coloring, job-shop and minimum cut problems. All of these solution methods are, however, efficient mainly for the specific problem class that they are meant to address. When the specific problem at hand deviates from the standardized class, the solution method typically becomes less efficient. The same is true also for compositional optimization, which exploits knowledge about the local behavior of subsystems to reduce the global state space.

Compositional SCT exploits the fact that if a certain sequence of events leads to something bad in one of the subsystems, then it is bad for the entire system and should be prevented. This makes sense since a correct execution of the full system requires all subsystems to function individually. In contrast to

Figure 2.4: A simple illustration of a railway system with two trains driving in opposite directions.

SCT, compositional optimization can not use such a simplification. There is no guarantee that the quickest or cheapest sequence of events in a subsystem is part of the quickest or cheapest sequence of events for the whole system. This can be illustrated with a simplified example of a railway system, shown in Fig 2.4. The example includes two trains A and B that are moving with the same speed in opposite directions from X to Y and from Y to X, respectively, and the global objective is to minimize their combined traveling time. It is easy to see that the optimal solution requires train A to wait at Z until B has arrived, since B otherwise would not be able to start until A has finished. If we, however, only optimize the path of A, without taking B into consideration, it would be more efficient to just continue directly without delays. This illustrates one of the main limitations of compositional optimization; there is typically not enough information available to decide on a local optimal path for each subsystem individually. The key challenge then becomes the identification of those parts of the subsystems that we do have enough information about to reduce locally.

2.3 CompOpt – an Optimal Compositional Optimization Technique

CompOpt, developed in Papers A and B, is one of the main contribution of this thesis. The technique is specifically designed for the mitigation of the state explosion problem during the optimization of large-scale systems of systems. It takes as input a modular optimization model where each subsystem is modelled individually as a *weighted automaton*, illustrated in Fig. 2.5, which include a cost function to specify the cost of each transition.

The main strengths of CompOpt are: (i) the identification and optimization of strictly local behaviour within each subsystem, and (ii) an integrated synchronization and optimization algorithm. These methods allow CompOpt to

compute global optimal solutions without computing any monolithic models.

Note that, although Paper B uses non-deterministic finite automata as the modelling paradigm, CompOpt is expecting deterministic systems. The non-deterministic models were only used in Paper B to express partial results generated by the integrated synchronization and optimization approach presented below, in which two transitions could have the same event but different weights. This does not affect the optimization algorithm, since the alternative paths are reduced down to one before the synchronization is done, thus resulting in a deterministic system.

Local optimization reduces each subsystem individually

The local optimization method is the core of CompOpt. We prove in Paper A that this method can compute maximal reductions of each subsystem or subproblem, called *locally optimal reductions*, without affecting the global optimal solution. The level of reduction is paramount to the complexity of the optimization, since it directly affects the extent of the state space growth in the subsequent synchronization.

The key to the local optimization method is to identify parts of the behavior in the subsystems that are strictly local, meaning that the behavior in these parts is independent of other subsystems. These parts can then be considered individually, without affecting the global problem.

To give a better understanding of the properties of local optimization, consider a small system consisting of two subsystems G_1, G_2. The task at hand is to optimize the first subsystem G_1 shown in Fig. 2.5(a), where the marking $\{\sigma, w\}$ of a transition indicates that the transition is activated by the event σ and has the weight w. The only available information about G_2 is that the event a is shared between the two subsystems in some way, how they interact is not revealed. The local optimization has to preserve the shared behavior, since this can affect the global behavior, which in this case means that the locally optimal reduction of G_1, denoted G_1', will include the same transition over event a. The rest of the behavior can be considered local and can, therefore, be optimized without affecting G_2. The resulting locally optimal reduction is shown in Fig. 2.5(b), where the event d is deactivated since it is more expensive than the sequence of local events b, c. The sequence of local events is abstracted to a single transition representing there combined behavior.

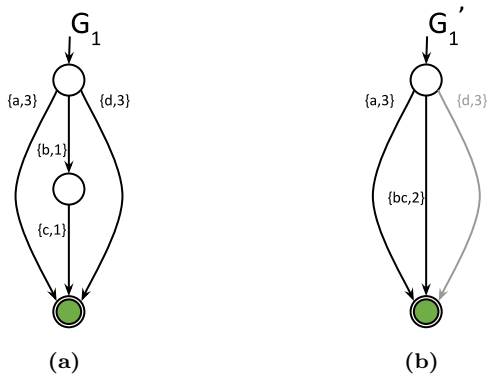


Figure 2.5: An illustration of the properties of local optimization. (a) shows a plant model of a subsystem G_1, where it is known that the event a is shared with another subsystem, (b) shows the locally optimal reduction of G_1, where the local transition $\{d, 3\}$ and the sequence $\{b, 1\}, \{c, 1\}$ have been replaced by an abstraction $\{bc, 2\}$.

Integrated synchronization and optimization

The integrated synchronization and optimization method called *partial time-weighted synchronization* (PTWS) was proposed for the first time in Paper B. This method mainly aims to reduce the complexity of dealing with a specific type of systems called time-weighted systems, where the cost function expresses the execution time of the tasks performed by each subsystem. CompOpt will then minimize the total execution time of the full behavior of the system, which requires synchronization of the time lines for the subsystems.

In SCT, this type of systems generally require more complex modelling paradigms such as *timed automata* [22]. The reason is that they no longer are pure DESs. The fact that the cost function or weights in these systems reflect the execution time of their transitions makes it necessary to also consider the synchronization of the execution of these transitions. This is something that by construction can be neglected for an ordinary DES, since the transitions are assumed to occur instantaneously. The synchronous composition will, by construction, model parallel events as a sequence. This makes sense in a DES, since the events are by definition instantaneous, and therefore can occur at the same time even if modelled sequentially. It does not make sense when the weights connected to the system transitions represent the execution time of

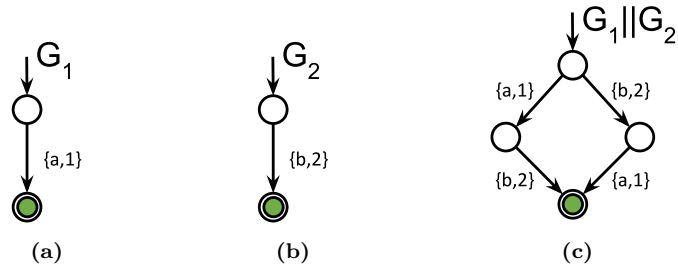


Figure 2.6: An example showing that default synchronous composition is insufficient when synchronizing time-weighted systems. (a-b) represent two independent subsystems that should run in parallel, (c) gives their synchronous composition.

the transitions, which would require a transition to finish its entire execution before it reaches the next state, where subsequent events can occur.

To illustrate this effect, consider the automata in Fig. 2.6, where (a) and (b) are two independent systems that run in parallel and (c) is the default synchronous composition of these systems. Let us first assume that the weight associated with each transition represents the energy consumption of the event. Then, a sequential model of the events a and b is correct. Each event, even if executed in parallel, still generates an individual energy consumption and the total consumption equals the sum of the two weights, which will be the result of the sequence of transitions in Fig. 2.6(c). Now let the weights represent the execution time of the transition. Since it is specified that the subsystems run in parallel we expect G_1 and G_2 to reach their marked state in 1 and in 2 seconds respectively, independent of each other. Following the behavior of the synchronous composition in Fig. 2.6(c) does, however, indicate that it will be the sum of these execution times that is required to reach the marked state. This shows that the default synchronous composition known from DES is insufficient for the synchronization of time-weighted subsystems.

To fully understand the type of parallel behavior that should be considered, take a look at the *timed Petri net*[23] in Fig. 2.7. The parallel behavior of the events a and b can easily be incorporated using a Petri net model since this paradigm has the ability to use multiple tokens, which in this case are used to represent that the two transitions are performed in parallel. The timed Petri net model is however a more complex model than an automaton,

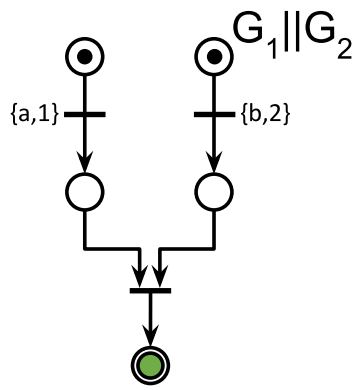


Figure 2.7: A timed Petri net model, representing a parallel execution of events a and b. Note that the final state is drawn as an accepting state only to highlight that it represents the same accepting state as in the corresponding automaton model. Generally, accepting states are not represented graphically for Petri nets.

including more information in each state about the current status of ongoing and possible transitions. To search through this system, one would still need to explicitly search through each variation of these states, which would have the same effect as using a timed automaton or any other more complex modelling paradigm.

An alternative approach to the more complex modelling paradigms described above is to apply simplifications to the time-weighted system, such as discretization of their timeline. This was done in Paper A by transforming all locally optimal reductions into *tick automata* [20] before their synchronization. In a tick automaton, each transition in the weighted automaton will be followed by a set of transitions – called ticks – which represent the passage of time. The number of ticks added to each transition depends on the desired accuracy. An example of this is illustrated in Fig. 2.8, where the subsystem G_1 from Fig. 2.5(a) has been transformed into a tick automaton with the accuracy of one time unit. This was sufficient to show the potential of CompOpt in general, but it was proven to be very inefficient with a reduced accuracy as well as a drastically increased state space of the optimization.

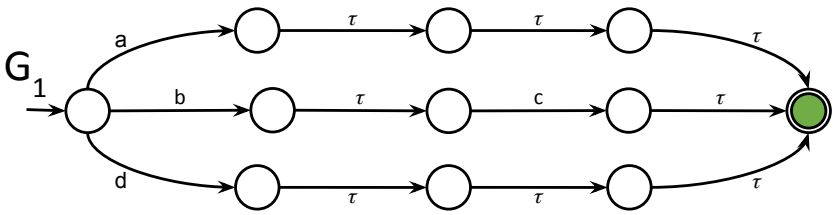


Figure 2.8: A tick automaton of the system depicted in Fig. 2.5(a) with the accuracy of one time unit.

Paper B addressed this limitation by proposing PTWS, denoted $||'$, which is able to model the parallel execution of tasks in multiple time-weighted systems using only regular weighted automata, such as in Fig. 2.6. Moreover, by integrating an optimization heuristic that resembles the local optimization, it can utilize the weights of the transitions in order to compute a synchronous composition that is partially reduced by construction. This enables the subsequent optimization of the composition to be faster due to the reduced search space.

To better understand the concept of PTWS, consider the weighted automaton in Fig. 2.9. This represents another synchronous composition of the modules in Fig. 2.6(a) and (b), which in this case has been constructed using PTWS. This example illustrates two of the main properties of PTWS. First, the weights of the transitions in the synchronous composition $G_1 \parallel' G_2$ are no longer directly related to the corresponding transitions in the subsystems G_1 and G_2. Instead, they have been modified during the synchronization in order to represent the parallel execution of the events. One can see that the weight of transition a is now set to zero. This means that the the transition is performed instantaneously, just like a regular DES, and the actual execution time of a is instead covered during the subsequent transition. The weight of b is kept at 2 to allow both event a and event b to finish their execution before the system ends up in the accepting state. Secondly, the model in Fig. 2.9 only includes a single path ab, in which the two events are executed in parallel. Thus, the model does not include any other possibility, such as, for instance, that the events are executed sequentially or that event b is executed before event a. This reduction is an effect of the included optimization heuristic. This is further explained in Paper B, but the basic idea is that it is enough

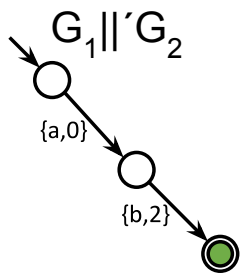


Figure 2.9: A time-weighted automaton representing the parallel execution of the events a and b.

to include only one path for any pair of local transitions, as long as this path is the cheapest. This allows PTWS to avoid the interleaving of local events.

Compositional computation of a global optimal solution

That CompOpt uses a compositional approach is stated as one of the main contributions in this thesis. But, once the local optimization and integrated synchronization and optimization are in place, the compositional computation actually becomes trivial.

From SCT we know that the compositional synthesis of a supervisor corresponds to an iterative process, where individual supervisors are computed for each subsystem and then combined incrementally, while doing further synthesis step-wise. This is the case also in CompOpt. For example, assume that set $G = \{G_1, G_2, \ldots, G_n\}$ represents a weighted system that should be optimized using CompOpt. Then an example of the iterative process of CompOpt can be described as

$$S'_i = \left(G'_i \parallel' S'_{i-1}\right)', \quad \forall i \in [1, n],$$

where G'_i represents the locally optimal reduction of a subsystem G_i and $\parallel'$ denotes synchronization using the integrated synchronization and optimization method presented above. For simplicity, this example synchronizes the subsystems in a strictly sequential order, but the algorithm presented in Paper B has a more general formulation that includes the subsystems in an arbitrary order. This allows CompOpt to utilize efficient heuristics to maximize the benefits of the compositional synthesis, such as [16].

Regardless of the ordering of the synchronization, this simple compositional approach guarantees that S'_n will be the final global optimal solution. The reason is that each iteration of the algorithm expands on the local behavior by synchronizing additional subsystems. Once the final model S'_n is reached, all tasks can be considered local, which will let this locally optimal reduction represent the global optimal solution.

Strengths and limitations of the approach

The strength of CompOpt is first of all that it fully integrates the optimization into a compositional framework. It is shown in Papers A and B that this has the potential to drastically reduce the state space of the optimization problem without computing large monolithic models.

However, from Section 2.2 we remember that one of the main limitations of any compositional optimization approach is that the information available when considering a single subsystem is typically insufficient for the computation of a final optimal control strategy. CompOpt is no exception to this rule. It does guarantee that the local optimization computes a minimal reduction of each subsystem, and that the final solution will be a global optima for the system, but we show in Papers A and B that the efficiency of CompOpt depends heavily on the complexity of the local behavior in the subsystems.

Taking the limitations into account, the results of the papers show that CompOpt has the potential to improve efficiency of the optimization of large-scale systems of systems, when these have a complex local behavior in the subsystems. Tab. 2.1 includes part of an example from Paper B, where a number of robots, in this case only 2, should each perform a varying number of tasks. The table indicates how the average size of the state space increases with the number of tasks to be performed. Without going into all the details about the example (these are fully described in Paper B), it is obvious that the exponential growth of this state space quickly becomes a problem for an optimization. In the paper, one can also observe that the growth becomes even worse when the number of robots is increased, instead of the number of tasks.

The results of solving the same systems as in Tab. 2.1 using CompOpt is shown in Fig. 2.10. Observe that the number of states are represented using a logarithmic scale. The figure originally comes from Paper B and also includes a varying number of robots, but the bottom blue line of the graph represents

Table 2.1: Average number of states in the monolithic model of an instance with 2 robots when the number of tasks is increased from 2 up to 10, calculated using instances with ten different random seeds.

Tasks	States
2	8 140.0
3	15 392.0
4	96 030.8
5	180 032.0
6	912 585.6
7	1 638 088.0
8	6 986 696.0
9	11 981 064.0
10	42 450 952.0

the case with two robots, which is equivalent to the example in the table. The figure shows that the state space no longer grows exponentially with an increased number of tasks. Instead, the growth resembles a low degree polynomial, since the line is almost linear. The paper further evaluates the potential of the approach, but also includes detailed examples that illustrate possible limitations of the optimization method.

Applying compositional optimization in industry

This thesis gives examples abstracted from industrial applications, both from logistics, in the motivating example of Section 2.1, and production industry, shown in the examples of Papers A and B. The compositional approach of CompOpt makes it especially relevant for large systems with multiple, mainly decoupled subsystems, with a limited shared behavior. The general formulation of CompOpt does, however, enable it to optimize any system of systems as long as these can be modelled using weighted automata. These type of systems can be found in a wide range of applications and are in no way restricted only to the traditional areas of industrial automation.

In the papers we apply CompOpt on an artificial example of a robot cell in production industry. This specific example may not be entirely realistic but there are similar scenarios in industry today. The example presented in these papers is a simplification of a respotting problem in a welding robot

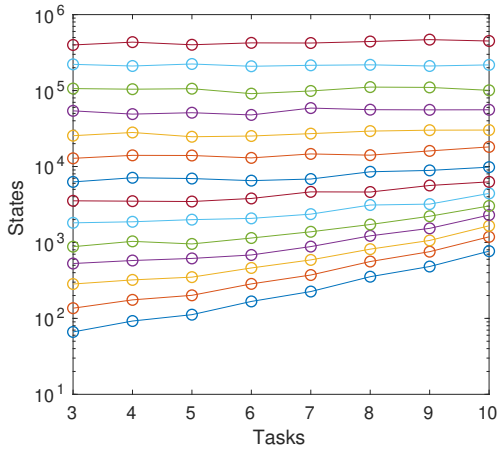


Figure 2.10: An example from Paper B, illustrating how the state space of CompOpt scales with the number of robots and tasks per robot. Each line in the plot represents a fixed number of robots ranging from 2 to 15.

cell. Just like the example, the real scenario includes multiple robots that operate in parallel on the same product, but from different angles. During a production cycle there are specific events that affect all robots similarly, such as the assembly of one additional subpart to the product. A few of the welding operations performed by the robots have to be performed while the assembly robot is still gripping the part. A majority of the operations, however, can, or have to, be performed once the assembly robot has left the zone.

CHAPTER 3

Modular active learning

The aim of this chapter is to introduce the modular active learning (MAL) proposed in Papers C, D and E, its purpose and properties.

3.1 Background

The areas of SCT and optimization tend to rely heavily on the existence of accurate models of the systems. There are plenty of well-known and efficient formal methods of verification, control and optimization that operate on a predefined plant model, CompOpt presented in Chapter 2 being one of them. Modeling large complex systems is, however, a challenging task that requires skill and in-depth knowledge of the system. Furthermore, manually defining these models is an error-prone task; incorrect or incomplete models are misleading, and can unnecessarily complicate the development process. Hence, assuming access to a correct plant model can be limiting.

Simulation technologies have gained attention in many areas of automation, and simulation-based development is increasingly common. In this area, the intended system is first developed in a simulation environment, where it can be tested and improved upon before moving forward with physical tests. These

simulations implicitly contain the behavior of the plant, though this behavior is often not accessible in a usable format for formal methods of verification or optimization. However, there exist learning algorithms that can infer the necessary discrete models of the plant by interacting with the simulation.

Active learning algorithms are a class of machine learning algorithms that aim to deduce a discrete-event model describing the behavior of a system. In active learning, the learning algorithm can query a teacher to label new data points. Active automata learning has been successfully applied to learn and verify communication protocols using Mealy machines [24], [25]; to obtain the formal models of biometric passports [26] and bank cards [27]. However, as mentioned in 1, these discrete-event models suffer from state explosion. Therefore, MAL was presented as a new modular approach to the learning of these models.

The *Modular Plant Learner* (MPL) introduced in Paper C was, to the best of our knowledge, the first attempt towards MAL within the automata learning community. This method learns complete modular models that can then be used in formal methods, for instance in verification and synthesis. The work on MAL was then extended in Paper D with the addition of a *Modular Supervisor Learner* (MSL). An algorithm that integrates SCT and active learning to learn a maximally permissive and controllable supervisor of the system directly from the simulation, given a set of formal specifications. Additionally, Paper E presented a modified version of MPL that computes a reduced plant model, specifically designed for use in CompOpt. The main strength of all three contributions are their ability to decrease the search space of the learning process by exploiting the modular structure of the system, thus mitigating the state explosion problem.

At the core of the MAL technique is the *plant structure hypothesis* (PSH), a high-level meta-model that defines the modular structure of the system. The modular structure refers to a division of the complete plant behavior as separate modules. Usually, but not necessarily, representing the separate subsystems that the plant is composed of. The PSH guides the learning algorithm to divide the learned information into separate modules and to reduce the search space, ultimately mitigating the state explosion problem.

Instead of viewing the system as a black box that must be modelled from scratch, the PSH assumes a certain amount of knowledge about the system. It is assumed that all actions that can be executed within the simulator are

known, but not the effect that these will have, or when they will occur. That is, the alphabet of the model is known but not the transitions. It is also assumed that all output variables are known in advance. A *state* in the simulator is defined by the values assigned to these output variables. Unique combinations of the values assigned to these variables make up the different states. The purpose of the PSH is to define the relation between these events and variables and the individual modules that together constitute the complete system model.

3.2 Defining the modular structure as a PSH

The PSH is formally defined as a 3-tuple $H = \langle M, E, S \rangle$, including a set of modules M and their event and state mappings $E(m)$ and $S(m)$, respectively. It is stated in Paper C that the event mappings define which actions, or events, that belong to which module. Thus, $E(m) \subseteq \Sigma$ is the local alphabet of module $m \in M$. Similarly, it is stated that the state mapping $S(m)$ defines the relation between the modules and the set of variables in the simulator.

In practice, these formal definitions do, however, only specify lower bounds for the mappings. Let us instead rephrase these definitions, saying that the event and state mapping of a module must include *at least* all events and states that are connected to the module. This simplifies the formulation of a correct PSH. Instead of answering the question "What events/state variables should be part of the mappings?", which might be hard without being an expert on the system, one can instead initiate the mappings to include all events/state variables and then ask "Are there any event/state variables that we *know* do *not* affect a specific module". This means that a correct PSH can be formulated regardless of the available system knowledge. Then, removing any event/state variable may speed up the learning greatly. If one manages to remove all events/state variables that are not strictly necessary, then a maximum reduction is achieved, both of the search space and of the state space in the final result.

The reformulation above has, of course, not changed the theory of the PSH. It aims only to help in the understanding of what the definition of a PSH actually requires and how to get started. Regardless of how the PSH is defined, provided that it is valid, the learning algorithm will return the same system. That is, a specific PSH affects how many modules the model will include and

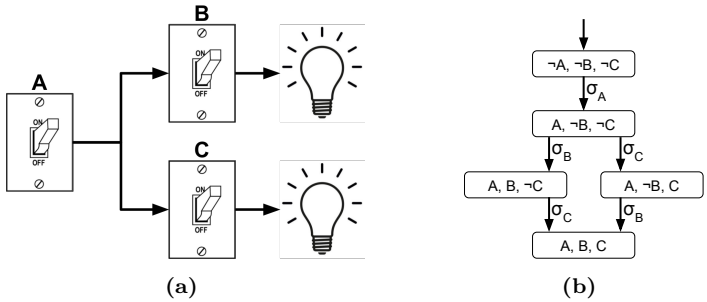


Figure 3.1: A system of systems consisting of two light bulbs controlled by separate switches as well as a main switch. (a) shows an illustration of the system, (b) shows the system model.

how much redundancy there will be between these modules, but the complete model, i.e. the synchronous composition, should always be the same. It should be noted though that the efficiency of the learning strongly correlates with the quality of the PSH; if none or very limited structural information is available, then MAL might not be the most effective learning method.

Example on the formulation of a PSH

Consider an illustrative example of a system consisting of two light bulbs and three switches, shown in Fig. 3.1(a). Switch A controls the main power to both lights and switches B, C control the power to one bulb each. For the simplicity of the example, the system also has the following restrictions: (i) the switches can only be toggled from OFF to ON, not the other way around and (ii) the main switch A has to be toggled ON before the switches B and C.

An automata representing the system is shown in Fig 3.1(b). Event label σ_i represents the toggling of switch i and each state in the model is a valuation of the state vector $\langle A, B, C \rangle$, where A, B and C are binary variables representing the status of the switches. To simplify the notation, we denote the valuation of these state variables as A when switch A is ON or $\neg A$ when it is OFF, and similarly for the state variables B and C.

Assume that the formal model, in Fig 3.1(b), is not yet known but that we have access to a simulator of the system and that we want to use MAL

to learn a modular model. The first step in the initiation of MAL is then to define the desired PSH, $H = \langle M, E, S \rangle$. For this purpose, the following basic properties of the system is known:

1. The purpose of the system is to control two light bulbs.
2. Three light switches control the bulbs, toggled on/off by the actions σ_A, σ_B and σ_C.
3. There are three available variables A, B and C that indicate the status of the switches.

Based on the first property the most obvious division of the system into modules is to learn a separate model for each light bulb. This gives the modules $M = \{L_1, L_2\}$, where we have given the two light bulbs arbitrary labels L_1, L_2. Following the steps above we then assign the trivial solution to the event and state mappings:

$$E(L_1) = E(L_2) = \{\sigma_A, \sigma_B, \sigma_C\}$$

$$S(L_1) = S(L_2) = \{A, B, C\}$$

Let us evaluate how different levels of expertise determine the possible reduction of this PSH and how it affects the learning.

No knowledge: As a general rule, modular learning is not recommended when there is no knowledge about how the events and variables connect to the modules. With no knowledge, there are no event or state variables to remove in the mappings of the PSH, which leaves us with the trivial solution. This means that there are no distinct differences between the defined modules and the learning will be unable to make any reductions to the modules. This makes the modular learning very inefficient and the final result will be that each module includes the complete system model, from Fig 3.1(b).

Knowledge about events: Assume that we know that the events σ_B and σ_C are connected to one light bulb each, and that we define the corresponding event mappings to be $E(L_1) = \{\sigma_A, \sigma_B\}$ and $E(L_2) = \{\sigma_A, \sigma_C\}$. This reduction of the event mapping will have a big impact on the modular model.

Consider for instance the effect on module L_1. That event σ_C is no longer part of the event mapping $E(L_1)$ implies that σ_C is not part of its alphabet

Σ_{L_1}, thus no transitions that are activated by this event can be part of the module. The goal of the modular learning will in this case be to learn a model where module L_1 only models the behavior including the events σ_A and σ_B, while module L_2 only models the behavior including the events σ_A and σ_C. The result will, for this example, be that each module includes only two transitions, while the interleaving of the events σ_B and σ_C, seen in 3.1(b), will be ignored. This shows that removing an event from an event mapping may reduce the size of the state space in the corresponding module.

Knowledge about state variables: Assume that we know that state variable C only connects to event σ_C. We can then reduce the state mapping to $S(L_1) = \{A, B\}$. This type of reduction to the state mapping lets MAL structure the identified states and transitions into the different modules directly, reducing the state explosion during the learning process. For instance, let us consider the two transitions activated by event σ_b. Given the reduction of the state mapping $S(L_1)$, the valuation of the source and target state are identical, i.e. both state variables in the state mapping have the same value in the source and the target state. From this, MAL infers that the two transitions corresponds to a single transition. Hence, querying the simulation for outgoing transitions using an event $\sigma \in E(L_1)$ from one of the target states can be expected to yield the same results as if it was done from the other target state. This shows that removing a state variable from a state mapping may reduce the search space of the learning.

To summarize, the PSH is used to incorporate structural knowledge about the system to the learning algorithm. Knowledge about the events may mitigate the state explosion of the final model by reducing the state space of each module, while knowledge about the state variables may mitigate the state explosion of the learning process itself.

3.3 The exploration and reduction of the search space

As mentioned previously, MAL is designed to conduct the learning through interactions with a simulation of the system, which provides several advantages compared to the physical system. A simulation can typically be run faster

than real-time, even multiple instances in parallel, thereby speeding up the learning process. Dangerous collisions and unforeseen events are avoided and confined to the simulation, providing a safe learning environment. This is not strictly necessary, however, and MAL could in practice interact with a physical system as well, as long as an interface exists.

The interface necessary to run MAL is defined by a function *getNextState* : $Q \times \Sigma \rightarrow Q$ that takes as input a state of the system and an event to be executed. The output of this function is the resulting state when the given event is executed from the given state. The learning then uses this algorithm in a forward search, starting in the initial state of the system, querying the simulator for unidentified outgoing transitions from any new state.

MAL is arguably the most complex process presented in this thesis, and with the aim to further improve the understanding, let us revisit the previous example with two light bulbs, shown in Fig. 3.1. Using MAL to learn a model of this system would yield a modular model, such as the one shown in Fig. 3.2. This specific modular model represents the output of the Modular Plant Learner (MPL), the most basic of the MAL algorithms, given the following PSH:

- $M = \{L_1, L_2\}$,
- $E(L_1) = \{\sigma_A, \sigma_B\}$,
- $E(L_2) = \{\sigma_A, \sigma_C\}$,
- $S(L_1) = \{A, B\}$,
- $S(L_2) = \{A, C\}$.

In the model of Fig 3.2, each module includes only the state variables and events connected to one of the light bulbs, and their synchronous composition equals the monolithic model in Fig. 3.1(b). The learning of this modular model is explained in three steps below.

Step 1.1 The learning starts in the initial state $q_0 = \langle \neg A, \neg B, \neg C \rangle$. The algorithm searches for outgoing transitions from this state through a number of queries to the simulator, namely one query $getNextState(q_0, \sigma)$ for each of the events in both $E(L_1)$ and $E(L_2)$. This yields a single transition $\langle \neg A, \neg B, \neg C \rangle \xrightarrow{\sigma_A} \langle A, \neg B, \neg C \rangle$, which can be seen in the Fig. 3.1(b). This is

Figure 3.2: A modular model of the system with two light bulbs, depicted in Fig. 3.1, with module L_1 in (a) and module L_2 in (b).

an example of a shared transition. It is activated by the event σ_A and updates the state variable A, both of which are part of the state and event mappings in L_1 as well as L_2. This tells MPL that the transition should be represented in both modules, resulting in the first transition of each module.

Step 1.2 The algorithm expands the newly identified state $\langle A, \neg B, \neg C\rangle$ from Step 1.1. Outgoing transitions are $\langle A, \neg B, \neg C\rangle \xrightarrow{\sigma_B} \langle A, B, \neg C\rangle$ and $\langle A, \neg B, \neg C\rangle \xrightarrow{\sigma_C} \langle A, \neg B, C\rangle$. These transitions are similar and each concerns only one of the modules. The first transition is activated by event σ_B and updates only state variable B, an event and a state variable that is only part of the event and state mapping of L_1. This means that the transition does not affect module L_2 at all and yields a new transition only in module L_1, namely the transition $\langle A, \neg B\rangle \xrightarrow{\sigma_B} \langle A, B\rangle$. Similarly, the second transition does not affect module L_1 and yields only the transition $\langle A, \neg C\rangle \xrightarrow{\sigma_C} \langle A, C\rangle$ in module L_2.

Step 1.3 Step 1.2 identified two new states: $\langle A, B, \neg C\rangle$ and $\langle A, \neg B, C\rangle$. We can already see in Fig. 3.2 that the local transitions added in Step 1.2 complete the modular model, but the algorithm does not know this yet and must search for any additional outgoing transitions. Additionally, this step, and the expansion of these states illustrate two very important properties of MAL.

First, even if the search finds a new transitions to an unidentified state, it does not necessarily yield any new transitions in the modular model. The reason is that the new state can represent an interleaving of previously identified local states, which does not need to be explicitly modelled in the mod-

ules, but will be available in their synchronous composition. Assume, for instance, that the state $\langle A, B, \neg C\rangle$ is expanded. This yields one new transition $\langle A, B, \neg C\rangle \xrightarrow{\sigma_C} \langle A, B, C\rangle$, with a previously unidentified target state. Its event and the only updated state variable is local to L_2, so it does not affect module L_1, and in L_2 the corresponding local transition $\langle A, \neg C\rangle \xrightarrow{\sigma_C} \langle A, C\rangle$ is identical to the transition added to L_2 in Step 1.2. Thus, the transition does not add anything to the modular model, and can therefore be ignored. This constitutes the main strength of a modular learning and may result in a drastic reduction of the search space when learning larger systems, where the modules typically have a large number of local variables that can interleave.

Secondly, the number of events that have to be evaluated in the expansion of a state may be reduced based on how the transition that lead up to this state affected different modules. This was first introduced in Paper E, and MAL had prior to this tried to expand each new state through every available event. To demonstrate this method, recall the transition $\langle A, \neg B, \neg C\rangle \xrightarrow{\sigma_B} \langle A, B, \neg C\rangle$ from Step 1.2, that lead up to the previously unidentified state $\langle A, B, \neg C\rangle$. In L_1, this added a new local state $\langle A, B\rangle$, which tells the algorithm that there might exist outgoing transitions from this state that must be included in the module.

The algorithm must search for these transitions by querying the simulator once for each event in the event mapping $E(L_1)$, which, by the definition of PSH, are the only events that can be included in module L_1. On the other hand, the transition did not affect module L_2. In fact, following the state mapping $S(L_2)$, we can see that the source and target states both correspond to the same local state $\langle A, \neg C\rangle$. From this, the algorithm infers that it is unnecessary to look for new transitions for module L_2, using the events in $E(L_2)$, since any such transition would have already been identified when expanding the source state. In other words, it is enough for the expansion of a state to consider events in the event mappings of those modules in which the identification of the state yielded a new local state.

Note that the above reduction of the number of events considered in the expansion of states relies on the the assumption that the state mappings of the PSH are always overestimated. While this allows the algorithm to decrease the number of queries to the simulator, which can have a big impact on the execution time of the learning process, it has its own side effect. If a state variable that affects module m is removed from $S(m)$, then the algorithm

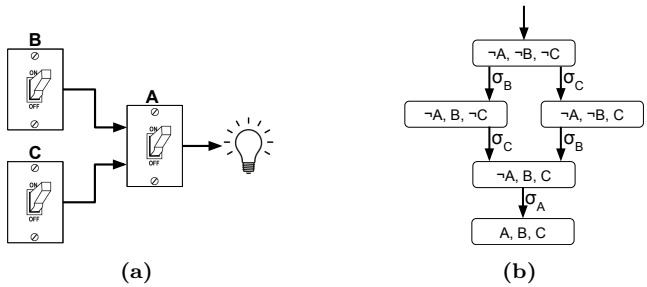


Figure 3.3: A system of systems consisting of three switches A, B and C that together control a single light bulb. (a) shows an illustration of the system, (b) shows the system model.

might ignore the expansion of some event in certain states, leading to an incomplete final model.

Coordinating shared transitions

Step 1.1 above shows how a shared transition is added to multiple modules. It is not, however, always trivial to identify these transitions in modular learning. To illustrate this, consider a new example, shown in Fig. 3.3, which includes a modified configuration of the three switches from the previous example. The purpose is to illustrate what happens when a shared event, i.e. switch A, depends on multiple local events, i.e. switches B and C. Previously A was a main switch that controlled the power to both B and C. Now switches B and C can be activated independently, but it is only once both are activated that switch A can be activated. This can, for instance, be a system where B and C are part of a two-step security check required before A can be activated.

Applying MAL to the system in Fig. 3.1 using the same PSH as previously would yield an incorrect result where the shared event would never be identified, which becomes obvious in the two steps described below.

Step 2.1 The learning still starts in the initial state $\langle \neg A, \neg B, \neg C \rangle$, but the expansion now yields two transition $\langle \neg A, \neg B, \neg C \rangle \xrightarrow{\sigma_B} \langle \neg A, B, \neg C \rangle$ and $\langle \neg A, \neg B, \neg C \rangle \xrightarrow{\sigma_C} \langle \neg A, \neg B, C \rangle$. These are similar to the transitions identified in Step 2 of the previous learning in the sense that they each add only a single local transition to one of the modules while leaving the other unchanged,

namely $\langle \neg A, \neg B \rangle \xrightarrow{\sigma_B} \langle \neg A, B \rangle$ to L_1 and $\langle \neg A, \neg C \rangle \xrightarrow{\sigma_C} \langle \neg A, C \rangle$ to L_2.

Step 2.2 Step 1 identified two states $\langle \neg A, B, \neg C \rangle$ and $\langle \neg A, \neg B, C \rangle$. Expanding these states yields two outgoing transitions $\langle \neg A, B, \neg C \rangle \xrightarrow{\sigma_C} \langle \neg A, B, C \rangle$ and $\langle \neg A, \neg B, C \rangle \xrightarrow{\sigma_B} \langle \neg A, B, C \rangle$. However, these transitions are, according to the given PSH, equivalent to those identified in Step 1. Consider, for instance, the first transition, which only affects module L_2 and, following $S(L_2)$, corresponds to the local transition $\langle \neg A, \neg C \rangle \xrightarrow{\sigma_C} \langle \neg A, C \rangle$. No new states are identified in this step and the algorithm believes that the system is explored completely. This means that the algorithm fails to identify the shared transition $\langle \neg A, B, C \rangle \xrightarrow{\sigma_A} \langle A, B, C \rangle$.

The problem above is that $S(L_1)$ includes B but not C, while $S(L_2)$ includes C but not B. This means that the algorithm interprets the state $\langle \neg A, B, C \rangle$ as an unnecessary interleaving of the two local states $\langle \neg A, B \rangle$ in L_1 and $\langle \neg A, C \rangle$ in L_2. To avoid this, the PSH needs to account for the coordination of the shared transition, which is done by including C in $S(L_1)$ and/or B in $S(L_2)$. Assume that we do both, i.e. $S(L_1) = S(L_2) = \{A, B, C\}$, while leaving the set of modules and the event mappings unchanged. This would then change the learning process in the way described in Step 3.1.

Step 3.1 This is similar to Step 2.1, but the two transitions now affect both modules, since the state mappings now include additional state variables. Consider, for instance, the first transition $\langle \neg A, \neg B, \neg C \rangle \xrightarrow{\sigma_B} \langle \neg A, B, \neg C \rangle$. Its effect on module L_1 is unchanged, yielding the local transition $\langle \neg A, \neg B \rangle \xrightarrow{\sigma_B} \langle \neg A, B \rangle$. However, in contrast to Step 2.1, the new PSH, which defines that $B \in S(L_2)$, implies that the target state is significant for the learning of module L_2. The problem is that $\sigma_B \notin E(L_2)$ implies that the transition can not be included in the final module. This means that, in module L_2, the state variable B should toggle from OFF to ON but no event is executed. This does not conform with a discrete event system, where all state changes must come as a result of a discrete event being activated. For now, assume that MAL adds the target state to the module without adding the transition, which is illustrated in Fig. 3.4. This means that the target state will be unreachable, but it will at least be included in the learning process and will be expanded. It will later in this subsection be shown how the unreachability of these states

Figure 3.4: The modular model so far after Step 3.1, with module L_1 in (a) and module L_2 in (b).

is resolved in the final modular model.

Step 3.2 Just as in Step 2.2, the learning identifies two outgoing transitions $\langle \neg A, B, \neg C \rangle \xrightarrow{\sigma_C} \langle \neg A, B, C \rangle$ and $\langle \neg A, \neg B, C \rangle \xrightarrow{\sigma_B} \langle \neg A, B, C \rangle$. However, in contrast to Step 2.2, the new PSH ensures that the target state $\langle \neg A, B, C \rangle$ is now recognized as an unidentified state and the corresponding local transitions are added to the modules. The result of this step can be seen in Fig. 3.5.

Step 3.3 Due to the new transitions in Step 3.2, the algorithm now correctly expands the state $\langle \neg A, B, C \rangle$ and adds the shared transition $\langle \neg A, B, C \rangle \xrightarrow{\sigma_A} \langle A, B, C \rangle$ to both modules.

The algorithm terminates after Step 3.3 when no additional transitions can be identified. At this point the algorithm must construct the final modular model based on the information learned and it must resolve the unreachability in the modules. This is done by, for each module m, removing any state variables from the state mapping $S(m)$ that were changed during the learning

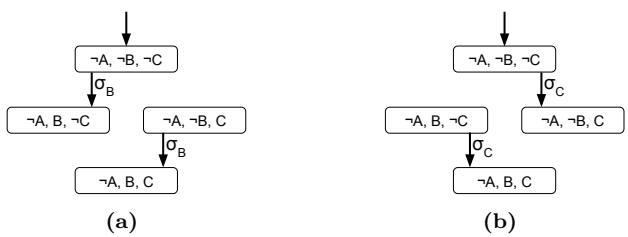


Figure 3.5: The modular model so far after Step 3.2, with module L_1 in (a) and module L_2 in (b).

Figure 3.6: The final modular model constructed after Step 3.3, with module L_1 in (a) and module L_2 in (b).

by a transition activated by an event that is not in the event mapping $E(m)$. Recall the transition $\langle \neg A, B, \neg C \rangle \xrightarrow{\sigma_C} \langle \neg A, B, C \rangle$, identified in Step 3.1. This transition updates the state variable $C \in S(L_1)$ even though $\sigma_C \notin E(L_1)$, which resulted in an unreachable state being added to the module. Removing C from $S(L_1)$ will change the structure of module L_1 by merging any pair of states where the only difference in their valuation is the value of C. For instance, the states $\langle A, \neg B, \neg C \rangle$ and $\langle A, \neg B, C \rangle$ both valuates to $\langle A, \neg B \rangle$ once variable C is removed from the mapping. Removing all such variables in both L_1 and L_2 will resolve the unreachability and yield the final modular model shown in Fig. 3.6.

3.4 Applying modular learning in industry

This chapter has introduced the basics of MAL, proposed in Papers C, D and E. The main purpose of this method is to enable formal methods to be used in verification, synthesis and optimization of large-scale automation systems, while relieving its requirement on expert system knowledge for the construction of the formal models. It applies a modular approach using the PSH – a much simpler meta-model of the system structure – to mitigate the state explosion, both during the learning and in the resulting modular model. This makes the method applicable to much larger systems than learning algorithms that do not take any structural information into account.

The key properties that determine the applicability of the method is: (i) the availability of a suitable simulator with which the learning can interact, (ii) existing knowledge about the structure of the system, i.e. the ability to define an efficient PSH, and (iii) the complexity of different subsystems in

the system at hand. The latter refers to the fact that, if the system includes multiple loosely connected systems, then there are likely a lot of unnecessary interleaving of local transitions that can be avoided in the modular model. However, if all subsystems work in close relation to each other and if they need to constantly keep track of each other, then there might not be a reason to apply a modular learning. In that case, all modules would mainly include the same shared transitions, possibly even worsening the state explosion problem.

Defining clearer guidelines to identify systems that benefit from a modular learning is hard and is left for future research. In general terms, the goal of modular learning is to avoid an exponential growth of the search space and of the state space of the constructed model. However, in worst case it will, when no local transition exist in the subsystems, result in a linear increase of the state space, since each module will have to include the complete system model.

CHAPTER 4

Compositional optimization of an unknown system

This chapter presents the integration of MAL and CompOpt, implemented in MOL in Paper E. Note, however, that MOL is an extension of MAL and some of its contributions, that do not directly relate to optimization, have already been covered in Chapter 3. The purpose of MOL is to construct the modular optimization models needed to apply CompOpt to the current system. This was proposed in Paper E to relieve CompOpt's strong reliance on predefined models. The main difference in these models compared to the models discussed in Chapter 3 is that a modular optimization model also must include a cost function, defining the cost of each transition. Paper E has focused on costs represented by the execution time of the transitions but it can, for instance, also be monetary costs or energy efficiency.

The goal of MOL is to integrate the local optimization of CompOpt directly in the learning algorithm. CompOpt can of course be applied on the modular models constructed by MPL as well, but the strength of MOL is that it learns locally optimal reductions of the modules. The result in Paper E shows that this has the potential to drastically reduce the state space of the constructed optimization model, which in turn decreases the complexity of the subsequent optimization. It should be noted that MOL does not, as of

yet, include the abstraction of sequences of local events that was illustrated in Fig. 2.5 in Chapter 2. This could of course be included, but would require a post-processing of already added transitions, thus might be more suited for the subsequent optimization.

The main strength of MOL, compared to MPL, is its ability to reduce the state space of the constructed modular model. This is done by guiding the search of the learning process to imitate a shortest-path algorithm. Similarly to the well-known Dijkstra's algorithm [28], MOL searches for the cheapest path from the initial state to all other states. This results in the learning of modules resembling shortest-path trees [29], which are then reduced by removing all paths that do not conform to the definition of a locally optimal reduction. Furthermore, MOL integrates this local optimization into an algorithm very similar to MPL, without adding any extra complexity. The only practical differences between MOL and MPL are: (i) that the order in which new transitions in the system is explored is decided by a greedy heuristic to ensure that the shortest path always gets expanded first, and (ii) that, when a new transition is identified it will be added to the modules only if no cheaper path to the target state has been previously identified.

4.1 Learning locally optimal reductions

Section 2.3 briefly described the computation of a locally optimal reduction as the maximal reduction of a subsystem without affecting the global optimal solution. It was stated that the key to this local optimization method is to identify parts of the behavior in the subsystems that are strictly local. In CompOpt this is done by identifying all shared transitions, which must be maintained in the modules. It then optimizes the local behavior of each subsystem by conducting one greedy search from the target state of each shared transition. Only the cheapest local paths connecting any pairs of states are kept.

The same method can not be implemented in MOL, since the shared transitions are not known in advance. Instead, the learning algorithm starts with a single greedy search from the initial state, identifying the cheapest local paths from this state to any other state. If the algorithm identifies a shared transition along the way, then it executes one additional greedy search from the target of this transition. In this way it will continuously expand the sys-

tem, identifying shared transitions along the way and optimal local paths to connect them, until, eventually, the complete system has been expanded.

4.2 Avoiding concurrent operations using MAL

Section 2.3 proposed PTWS as a method to compute a synchronous composition of multiple time-weighted systems. A modelling paradigm that is used in CompOpt to optimize the system based on the execution time of the transitions. The purpose of PTWS is to deal with the challenge of explicitly modelling concurrent operations of tasks in parallel time-weighted systems. This challenge also applies to the learning of these systems, where constructed models must include the same concurrent operations.

Fortunately, just like in CompOpt, this challenge only arises when multiple subsystems operating in parallel should be included in the same model. Thus, it can be ignored completely when learning modular models, assuming that the internal operations in each module are executed sequentially. This is an implicit assumption in MAL, since it models each module as a deterministic finite automaton [7]. This is a modelling paradigm that does not include the execution of multiple actions in parallel, therefore, MAL implicitly expects each module to execute sequentially. If the system does include subsystems that operate in parallel, then the concurrent operation of their tasks will still be included in the synchronous composition of the modules.

In summary, using a modular learning avoids certain challenges that arise in connection with explicit modelling of concurrent operations in time-weighted systems. This is an indirect strength in MOL, since it allows for much more efficient learning of optimization models for this type of systems.

4.3 Completing the compositional optimization framework

While MOL is presented as a separate contribution; nonetheless, it is specifically designed to learn modular optimization models that can be solved by CompOpt. Thus, MOL can be considered a complement to the CompOpt framework, with the ability to simplify its application.

Previously, applying CompOpt on an automation system first required the

formulation of a formal modular model of the system. Formulating these models requires a high level of expertise and system knowledge. With the addition of MOL, it is now sufficient to have access to a simulation of the system and to formulate the much simpler PSH. In practice, the process of applying the combination of MOL and CompOpt is this:

1. Construct a simulation of the system. This can be implemented in any software as long as a suitable interface exists or can be constructed.

2. Define a PSH for the system. As shown in Chapter 3, this can theoretically be constructed without any system knowledge, but the more structural information that is added to PSH, the more efficient the learning and subsequent optimization will be.

3. Let MOL construct a modular optimization model based on the simulation and PSH.

4. Input the constructed model to CompOpt in order to compute the global optimal solution.

4.4 Applications in industry

This chapter has introduced the basics of MOL, proposed in Paper E. The method integrates the local optimization of CompOpt with the modular learning of MAL, to learn a modular model consisting of the locally optimal reduction of the subsystems. The purpose of MOL is to resolve the main limitation to the industrial application of CompOpt, namely its reliance on predefined formal models. Combining MOL and CompOpt completes an optimization framework that allows for optimization of large-scale automation systems, while minimizing the level of expertise and system knowledge needed for its application.

The key properties that determine the applicability of MOL is, just like with MAL in general: (i) the availability of a simulator, (ii) the ability to define an efficient PSH, and (iii) the complexity of different subsystems in the system at hand. It should be noted that MOL, in general, exploits the same local behavior of the subsystems as CompOpt, thus does especially well when learning the same type of systems as CompOpt is designed to optimize.

CHAPTER 5

Summary of included papers

This chapter provides a summary of the included papers.

5.1 Paper A

Fredrik Hagebring, Bengt Lennartson
Compositional optimization of discrete event systems
Proceedings of the *14th IEEE Conference on Automation Science and Engineering*,
2018, Munich, Germany.

This paper presents CompOpt as a novel optimization technique that integrates techniques from compositional supervisory control with traditional graph based search algorithms. Its strength comes from the ability to reduce the state space of each subsystem individually by exploiting their local behavior, mitigating the state explosion that would otherwise occur during synchronization. The technique shows great potential in dealing with large-scale systems of systems.

5.2 Paper B

Fredrik Hagebring, Bengt Lennartson
Time-optimal control of large-scale systems of systems using compositional optimization
Discrete Event Dynamic Systems: Theory and Applications,
Vol. 29 Issue 3 pp. 411-443, Sep. 2019.

This paper improves on CompOpt by proposing a novel and efficient synchronization method for time-weighted systems, called *reduced asynchronous synchronization* (RAS). This method is able to synchronize the parallel behaviour of time-weighted subsystems without adding any additional states or transitions to their models. The key is the integration of an optimization heuristic that, similarly to the local optimization, reduces the state space of the synchronous composition by removing non-optimal or redundant solutions, while maintaining the global optimal solution. We show in this paper that this further improves the efficiency of CompOpt by strengthening the mitigation of the state explosion problem.

5.3 Paper C

Ashfaq Farooqui, **Fredrik Hagebring**, Martin Fabian
Active learning of modular plant models
Proceedings of the *15th IFAC Workshop on Discrete Event Systems*,
2020, Virtual Conference.

In this paper we present the Modular Plant Learner (MPL), an algorithm that explores the state-space and constructs a discrete model of a system. The MPL takes as input a hypothesis structure of the system – called the *Plant Structure Hypothesis* (PSH)– and using this information, interacts with a simulation of the system to construct a modular discrete-event model. Using an example we show how the algorithm uses the structural information provided – the PSH – to search the state-space in a smart manner, mitigating the state-space explosion problem.

5.4 Paper D

Fredrik Hagebring, Ashfaq Farooqui, Martin Fabian
Modular supervisory synthesis for unknown plant models using active learning
Proceedings of the *15th IFAC Workshop on Discrete Event Systems*, 2020, Virtual Conference.

This paper proposes an approach to synthesize a modular discrete-event supervisor to control a plant, the behavior model of which is unknown, so as to satisfy given specifications. To this end, the Modular Supervisor Learner (MSL) is presented that based on the known specifications and the structure of the system defines the configuration of the supervisors to learn. Then, by actively querying the simulation and interacting with the specification it explores the state-space of the system to learn a set of maximally permissive controllable supervisors.

5.5 Paper E

Fredrik Hagebring, Ashfaq Farooqui, Martin Fabian, Bengt Lennartson
On optimization of automation systems: integrating modular learning and optimization
Submitted for possible publication in *IEEE Transactions on Automation Science and Engineering: Special Issue on Automation Analytics beyond Industry 4.0: From Hybrid Strategy to Zero-Defect Manufacturing*, 2021.

This paper extends on MPL with the Modular Optimization Learner (MOL), a method that learns modular optimization models designed for subsequent optimization using CompOpt. This completes the CompOpt framework, which allows for optimization of large-scale automation systems without the existence of formal models. Furthermore, an integrated greedy search heuristic removes many suboptimal paths in the individual modules. This allows MOL to construct reduced optimization models similar to the locally optimal reductions from CompOpt, speeding up the subsequent optimization.

CHAPTER 6

Concluding Remarks

The inspiration for the research presented in this thesis was an evaluation of existing optimization paradigms where we tried to identify efficient methods to deal with systems of systems. The evaluation did, however, result in the realization that all the evaluated methods suffered severely from the state explosion problem. This was, of course, expected, since the problem is caused by the modelling rather than the optimization itself. The aim has since then been to push the boundaries of large-scale optimization of systems of systems through the integration of modular techniques, which have proven very efficient in areas such as verification and supervisory control synthesis.

The first main contribution of this thesis, presented in Chapter 2, is a novel approach to optimization of large-scale systems of systems. This method, called compositional optimization, integrates time optimal control with methods from compositional supervisory control. The three key components of this method are: (i) a local optimization technique that reduces the size of each subsystem individually to mitigate the state explosion problem, (ii) an integrated synchronization and optimization technique that synchronizes the behavior of multiple subsystems, and at the same time reduces the global state space using a fully integrated optimization heuristic, and (iii) the composi-

tional approach that computes the global optimal solution of the complete system using the results from (i) and (ii). It is proven in this thesis that the proposed compositional optimization approach both maintains the global optimal solution of the system and computes a minimum reduction of each subsystem.

It is shown in the included papers that this method has the potential to be very efficient in large-scale optimization. Moreover, it is shown that it can scale very well with the number of subsystems. This is especially true when the subsystems have a complex local behavior, something that in a monolithic optimization would cause an exponential growth of the search space. It is shown in this paper that the method can calculate globally optimal solutions for large-scale industrial applications. The focus has mainly been on automation systems, including examples of manufacturing and logistics systems. Yet, the general theories presented can be applied to any system of systems, as long as it can be modelled as a discrete event system.

The second main contribution of the thesis, presented in Chapter 3, is a modular learning approach. Based on the interaction with a simulator and an initial hypothesis of the modular structure of the system, this approach autonomously learns the modular model necessary to perform a compositional optimization. This mitigates the main limitation of the compositional optimization: the reliance on existing formal models of the system, which requires a high level of expertise to construct. It is also shown that the modular learning is able to learn different types of models for multiple purposes, such as verification, supervisory control synthesis, as well as compositional optimization.

Chapter 4 shows how these two contributions can be combined, completing an optimization framework that can compute globally optimal schedules or plans for large-scale systems without prior knowledge of any explicit system models. This allows the optimization to be applied in a wider range of industrial systems to improve their overall efficiency, which, of course, is beneficial both from an economic and a sustainable perspective.

In future work, it would be interesting to apply this method in real industrial applications to evaluate how well the learning scales with different types of simulators and systems. The main challenge that this thesis reveals is to find better guidelines on how to define an efficient PSH, such that the local behavior of the subsystems can be exploited maximally. In the long term, the

definition of an efficient PSH can potentially be automated in the simulation software. That is, the simulation software could be extended with a function that allows it to derive a PSH based on the specific subsystem included in the simulation environment. For instance, if the simulation includes two industrial robots, then the simulator should be able to derive what actions can be performed by each robot and how this affects different output variables of the simulation. Furthermore, it would also be of interest to implement parallel computation of the subproblems in both the learning process and the compositional optimization, and to implement this as a cloud service to evaluate the potential of having scalable computational power.

References

[1] M. P. Groover, *Fundamentals of modern manufacturing : materials, processes and systems.* Englewood Cliffs, N.J. : Prentice Hall, cop. 1996., 1996, ISBN: 0-13-312182-8.

[2] K. M. Passino and P. J. Antsaklis, "On the optimal control of discrete event systems," in *Proceedings of the 28th IEEE Conference on Decision and Control*, Dec. 1989.

[3] B. A. Brandin and W. M. Wonham, "Supervisory control of timed discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 39, no. 2, pp. 329–342, Feb. 1994.

[4] J. Huang and R. Kumar, "Optimal nonblocking directed control of discrete event systems," *IEEE Transactions on Automatic Control*, vol. 53, no. 7, pp. 1592–1603, Aug. 2008.

[5] A. Kobetski and M. Fabian, "Time-optimal coordination of flexible manufacturing systems using deterministic finite automata and mixed integer linear programming," *Discrete Event Dynamic Systems*, vol. 19, no. 3, pp. 287–315, Sep. 2009.

[6] F. Hagebring, O. Wigström, B. Lennartson, S. I. Ware, and R. Su, "Comparing MILP, CP, and A* for multiple stacker crane scheduling," in *13th International Workshop on Discrete Event Systems (WODES)*, May 2016, pp. 63–70.

[7] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems, 2nd Ed.* Springer Science & Business Media, 2008.

[8] A. Valmari, "The state explosion problem," in *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the Volumes Are Based on the Advanced Course on Petri Nets*, 1998.

[9] S. I. Gass and M. C. Fu, *Encyclopedia of Operations Research and Management Science, 2013 Ed.* Springer US, 2013.

[10] W. B. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. New York, NY, USA: Wiley-Interscience, 2007, ISBN: 0470171553.

[11] X. Cao, *Stochastic Learning and Optimization: A Sensitivity-Based Approach*. Springer US, 2007, ISBN: 9780387367873.

[12] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-dynamic programming*. Belmont, MA, USA: Athena Scientific, 1996.

[13] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 3rd. Belmont, MA, USA: Athena Scientific, 2005, vol. I.

[14] K. C. Wong and W. M. Wonham, "Modular control and coordination of discrete-event systems," *Discrete Event Dynamic Systems*, vol. 8, no. 3, pp. 247–297, Oct. 1998.

[15] P. Ramadge and W. Wonham, "Modular feedback logic for discrete event systems," *IFAC Proceedings Volumes*, vol. 20, no. 9, pp. 93–98, 1987.

[16] H. Flordal and R. Malik, "Compositional verification in supervisory control," *SIAM Journal on Control and Optimization*, vol. 48, no. 3, pp. 1914–1938, 2009.

[17] S. Mohajerani, R. Malik, and M. Fabian, "A framework for compositional synthesis of modular nonblocking supervisors," *IEEE Transactions on Automatic Control*, vol. 59, no. 1, pp. 150–162, Jan. 2014.

[18] R. Hill and S. Lafortune, "Planning under abstraction within a supervisory control context," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, Dec. 2016.

[19] ——, "Scaling the formal synthesis of supervisory control software for multiple robot systems," in *2017 American Control Conference (ACC)*, May 2017.

[20] S. Ware and R. Su, "Time optimal synthesis based upon sequential abstraction and its application to cluster tools," *IEEE Transactions on Automation Science and Engineering*, vol. 14, no. 2, pp. 772–784, Apr. 2017.

[21] J. van Leeuwen, Ed., *Handbook of Theoretical Computer Science (Vol. A): Algorithms and Complexity*. Cambridge, MA, USA: MIT Press, 1990, ISBN: 0-444-88071-2.

[22] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994, ISSN: 0304-3975.

[23] R. David and H. Alla, *Discrete, Continuous, and Hybrid Petri Nets*. Jan. 2010.

[24] B. Steffen, F. Howar, and M. Merten, "Introduction to active automata learning from a practical perspective," in *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, 2011.

[25] B. Jonsson, "Learning of automata models extended with data," in *Formal Methods for Eternal Networked Software Systems: 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM 2011, Bertinoro, Italy, June 13-18, 2011. Advanced Lectures*. Berlin, Heidelberg: Springer, 2011, pp. 327–349.

[26] F. Aarts, J. Schmaltz, and F. Vaandrager, "Inference and abstraction of the biometric passport," in *Leveraging Applications of Formal Methods, Verification, and Validation*, T. Margaria and B. Steffen, Eds., Berlin, Heidelberg: Springer, 2010, pp. 673–686, ISBN: 978-3-642-16558-0.

[27] F. Aarts, J. de Ruiter, and E. Poll, "Formal models of bank cards for free," *IEEE 6th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2013*, pp. 461–468, Mar. 2013.

[28] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959, ISSN: 0029-599X.

[29] R. Sedgewick and K. D. Wayne, *Algorithms*. Addison-Wesley, 2011, ISBN: 9780321573513.