Thesis for the degree of Doctor of Philosophy

Energy efficient multi-robot coordination

Oskar Wigström

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2016

Energy efficient multi-robot coordination
Oskar Wigström
ISBN 978-91-7597-502-3

Doktorsavhandlingar vid Chalmers tekniska högskola
Ny serie nr 4183
ISSN 0346-718X

Department of Signals and Systems
Chalmers University of Technology
SE–412 96 Göteborg, Sweden
Telephone + 46 (0) 31 – 772 1000

Typeset by the author using LaTeX.

Printed by Chalmers Reproservice
Göteborg, Sweden 2016

To those left behind

Abstract

When multiple systems work in the same physical environment, it is important to ensure that no collisions occur. This thesis is focused on the centralized offline coordination of such collaborating systems, with the condition that the spatial path each system travels along is known before hand. In addition to collisions, dynamic constraints as well as optimization of a performance criterion are considered.

The problem is decomposed into two parts, a sequencing problem and coordination subproblem. For the sequencing problem, an algorithmic improvement is proposed, where constraint propagation methods from the computer science community are introduced to improve existing mixed integer nonlinear programming methods used in mathematical programming. The coordination subproblem on the other hand is approached from a modeling perspective. By applying state space discretization and variable changes, two models are derived, one which is entirely convex. Also, a two stage abstraction approach is introduced, where dynamic programming is used to parameterize part of the problem, resulting in a much simpler model at the next stage.

The above methods can be used for minimum energy coordination of industrial robots. Experimental results from the two robot case study are presented. In addition, the one robot case is also studied, where the execution time, robot payload and minimization criteria are varied. Furthermore, the application of the presented methods to hybrid systems is also discussed.

Finally, the slightly different problem of minimum time stacker crane scheduling is considered. In the stacker crane problem, a number of tasks should be allocated to a set of robots moving along the same one dimensional track. Although the exact spatial path is unknown in the stacker crane problem, it is shown that in a minimum time setting, it is still possible to use state space discretization.

Keywords: Optimization, scheduling, multi-robot systems, hybrid systems, energy minimization.

Acknowledgments

As my dissertation draws ever closer, I have become increasingly aware of its significance. A chapter of my life is ending, and a new one about to begin. Nostalgia and anticipation are ever present. Whatever comes next, I'd like to take a moment acknowledge some of the people who have starred in this latest chapter.

My supervisor, Bengt Lennartson, the never tiring professor, who would probably feel lost without a deadline to push. I offer you my sincerest gratitude for all your help during these years. Your primary contribution has been to teach me by example, the art of seeing possibilities, where there seemingly are none.

The crack team of administrators working at the department, whom without nothing would ever get done. In particular Christine, Elin, Lars, Natasha and Madeleine, you have helped make my life at Chalmers much easier.

Present and formers members of the Automation group, and in particular Martin, the boss-friend every employee would like to have, and Sarmad, my ever supportive colleague. Also Nikolce and Fredrik, I truly enjoyed our collaborations. And finally, special thanks to Sahar, Zhennan, Julia and Anna-Maria, for all the laughs and good talks.

To all of my friends outside of Chalmers, it makes me glad to say that you are far too many to mention individually. I deeply appreciate our friendship and would like to thank you from the very bottom of my heart, for being there.

I'm grateful to my family, mom, dad, my brothers and of course Anna and Edith. Also my extended family, whom have always made me feel welcome during these past years. And let's not forget my Chalmers family, Nina, Malin and Thomas, I hope that our lives will forever be intertwined.

My other half, Elin, you are what's most important to me. Every day, you change me for the better.

Oskar Wigström
Göteborg, October 2016

List of publications

This thesis is based on the following appended papers:

Paper 1. Oskar Wigström, Nikolce Murgovski, Sarmad Riazi and Bengt Lennartson. *Computationally efficient energy optimization of multiple robots.* Submitted for possible journal publication.

Paper 2. Oskar Wigström, Bengt Lennartson, Alberto Vergnano and Claes Breitholtz *High-level scheduling of energy optimal trajectories.* IEEE Transactions on Automation Science and Engineering, 10 (1), 57-64, 2013.

Paper 3. Oskar Wigström, Sarmad Riazi and Bengt Lennartson. *Constraint programming and nonlinear scheduling.* Submitted for possible journal publication.

Paper 4. Sarmad Riazi, **Oskar Wigström**, Kristofer Bengtsson and Bengt Lennartson *Energy and peak-power optimization of time-bounded robot trajectories.* Conditionally accepted for publication in IEEE Transactions on Automation Science and Engineering.

Paper 5. Oskar Wigström and Bengt Lennartson *An integrated CP/OR method for optimal control of modular hybrid systems* IFAC Workshop on Discrete Event Systems, 47 (2), 485-491, 2014.

Paper 6. Fredrik Hagebring, **Oskar Wigström**, Bengt Lennartson, Simon Ian Ware and Rong Su *Comparing MILP, CP, and A* for multiple stacker crane scheduling.* International Workshop on Discrete Event Systems, 63-70, 2016.

Other relevant publications co-authored by Oskar Wigström:

Nina Sundström, **Oskar Wigström** and Bengt Lennartson *On the conflict between energy, stability and robustness in production schedules* IEEE Conference on Automation Science and Engineering, 2016.

Sarmad Riazi, Kristofer Bengtsson, Rainer Bischoff, Andreas Aurnhammer, **Oskar Wigström** and Bengt Lennartson *Energy and peak-power optimization of existing time-optimal robot trajectories* IEEE Conference on Automation Science and Engineering, 2016.

Bengt Lennartson, Kristofer Bengtsson, **Oskar Wigström** and Sarmad Riazi *Modeling and optimization of hybrid systems for the tweeting factory* IEEE Transactions on Automation Science and Engineering 13 (1), 191-205, 2016.

Bengt Lennartson, **Oskar Wigström**, Sarmad Riazi and Kristofer Bengtsson *Modeling and optimization of hybrid systems* Conference on Analysis and Design of Hybrid Systems 48 (27), 351-357, 2015.

Bengt Lennartson, Kristofer Bengtsson and **Oskar Wigström** *Optimization of hybrid Petri nets with shared variables* IEEE Conference on Automation Science and Engineering, 1395-1396, 2015.

Sarmad Riazi, Kristofer Bengtsson, **Oskar Wigström**, Emma Vidarsson, Bengt Lennartson *Energy optimization of multi-robot systems* IEEE Conference on Automation Science and Engineering, 1345-1350, 2015.

Bengt Lennartson, **Oskar Wigström**, Martin Fabian and Francesco Basile *Unified model for synthesis and optimization of discrete event and hybrid systems* IFAC Workshop on Discrete Event Systems, 47(2), 86-92, 2014.

Sarmad Riazi, Payam Chehrazi, **Oskar Wigström**, Kristofer Bengtsson, Bengt Lennartson *A gossip algorithm for home healthcare scheduling and routing problems* IFAC World Congress, 47 (3), 10754–10759, 2014.

Oskar Wigström and Bengt Lennartson *Towards integrated OR/CP energy optimization for robot cells* IEEE International Conference on Robotics and Automation, 1674-1680, 2014.

Oskar Wigström and Bengt Lennartson *Integrated OR/CP optimization for Discrete Event Systems with nonlinear cost* IEEE Conference on Decision and Control, 7627-7633, 2013.

Oskar Wigström and Bengt Lennartson *Sustainable production automation-energy optimization of robot cells* IEEE International Conference on Robotics and Automation, 252-257, 2013.

Riazi Sarmad, **Oskar Wigström**, Seatzu Carla and Bengt Lennartson *Benders/gossip methods for heterogeneous multi-vehicle routing problems* IEEE Conference on Emerging Technologies and Factory Automation, 1-6, 2013.

Oskar Wigström, Nina Sundström and Bengt Lennartson *Optimization of hybrid systems with known paths* IFAC Conference on Analysis and Design of Hybrid Systems, 45 (9), 39-45, 2012.

Oskar Wigström and Bengt Lennartson *Scheduling model for systems with complex alternative behaviour* IEEE Conference on Automation Science and Engineering, 587-593, 2012.

Nina Sundström, **Oskar Wigström**, Petter Falkman and Bengt Lennartson *Optimization of operation sequences using constraint programming* IFAC Symposium on Information Control Problems in Manufacturing, 45 (6), 1580-1585, 2012.

Oskar Wigström and Bengt Lennartson *Energy optimization of trajectories for high level scheduling* IEEE Conference on Automation Science and Engineering, 654-659, 2011.

List of acronyms

BB	–	Branch and Bound
GDP	–	Generalized Disjunctive Program
LP	–	Linear Program
MINLP	–	Mixed Integer Nonlinear Program
MIQCQP	–	Mixed Integer Quadratically Constrained Quadratic Program
MP	–	Master Problem
MSOCP	–	Multi Stage Optimal Control Problem
NLP	–	Nonlinear Program

Contents

Part I

Introductory chapters

Chapter 1

Introduction

This work concerns the coordination of multiple robots moving in a shared physical environment. Of particular interest are the dynamics of robotic manipulators and the use of state space discretization methods. Throughout the upcoming chapters, we will attempt to answer the following three questions. How can we pose a mathematical programming model for the coordination problem, including enough detail to bound acceleration and minimize energy and/or time? Is it possible to augment existing mathematical programming algorithms by exploiting the specific structure of the coordination problem? And what are the potential energy savings for robotic manipulators using the developed methods?

The thesis consists of two parts. Part I unifies the material found in the appended papers and Part II contains the appended papers. Some of the material found in the appended papers is also extended in Part I.

1.1 Scope

The models and algorithms in this thesis all utilize state space discretization in some form. Hence, to enable state space discretization, we will restrict the system dynamics to that of a single monotonically increasing position state. By use of position varying bounds on velocity and acceleration, we will still be able to model higher dimensional acceleration bounded systems moving along fixed paths. The two exceptions to the monotonicity condition are a discussion on hybrid systems and a stacker crane scheduling problem addressed towards the end of the thesis.

As for model parameters, we assume that each system is defined by the fixed path it moves along, and the velocity and acceleration bounds it must stay within. Detailed system parameters which could be used to express for example torque are assumed to be unknown.

Regarding problem size, we have chosen to regard the coordination problem from an open loop planning perspective, as this is the setting in which robotic manipulators are often scheduled. That is, our problem is neither real time critical, nor is it subject to any disturbances. The solution times should preferably lie in the range of a seconds up to a few minutes. Within this time frame we would like to solve as large problem instances as possible.

Recall that the robots are to share a physical environment, and as such, there should be some mechanism to describe interaction between systems. In thesis thesis, we will focus on avoiding collisions. That is, for each pair of systems, there exists a set of forbidden position states, where collision occurs. A feasible solution should be collision free.

1.2 Background

Energy efficiency is a very important design driver for robots and other moving devices in manufacturing systems. Due to the energy cost of operating industrial robots, manufacturing industries are concerned with decreasing their electricity consumption. For instance, in a typical body shop, over 500 robots assemble roughly 300 to 500 parts using a total of 3500 to 5000 spot welds before they are dispatched to the paint shop [1], [2]. An average 200 kg payload body shop robot consumes a yearly 8 MWh [3], and robots overall consume approximately 8% of the total electrical energy in automotive production [4]. Also new policies for energy efficiency are incentives for manufacturing industries to reduce electricity consumption [5].

There exist several strategies for increased energy efficiency in industrial manufacturing. One approach is equipment selection during the process design stage, such as type of robots [6], motor and/or gearhead [7], [8]. Another approach is changing the controller in order to minimize a certain criterion, such as the integral squared torque [9] or power [10], at the cost of execution time. A drawback with these methodologies is that they are intrusive, i.e. require changes to configuration or changes in production rate.

In contrast, a non-intrusive strategy, such as the trajectory planning of multiple robots, can reduce energy without affecting the production rate, by efficient utilization of idle time. Two approaches are presented in [11], where idle time between the operations is used to reduce velocities and accelerations, although without explicitly considering the energy consumption. Another approach is [12], where the energy use for individual robot tasks is parametrized, and incorporated into a mixed integer nonlinear scheduled model.

The work in this thesis began in 2010 by extending the parameterization approach in [12], resulting in Paper 2. Last year, research on energy consumption in industrial robots was surveyed in [13], and amongst the body of studies, only a few had addressed non-intrusive energy optimization. In fact, the parameterization approach in [12], a practical implementation of said approach [14], and the extension in Paper 2, were the only non-intrusive studies concerned with energy reduction through scheduling.

Another neighboring field is robotics. A recent survey on multiple mobile robot systems [15] would categorize our problem formulation within motion planning, which is further subdivided into: cell decomposition, potential field and roadmap approaches. All of the cited approaches reduce the continuous motion planning problem to a graph search problem, whereas in this thesis we apply a mathematical programming approach. Our work also differs from these in that we consider more detailed dynamics and focus on energy minimization under fixed cycle time. However, this work is limited by the assumption that the path is fixed. One notable exception

within the field of robotics is the mathematical programming model presented in [16], which concerns the time minimal problem with mobile robot dynamics.

Our robot coordination problem is also somewhat similar to intelligent transportation systems. Linear criteria include the average and maximum vehicle delay [17], which are similar to due dates and cycle time in robotics and robot scheduling. Much like minimum energy or jerk formulations for robots, there are dynamic criteria such as velocity tracking and actuator/discomfort penalties for transportation systems, e.g. as in [18]. There is naturally a somewhat greater emphasis on guaranteeing safety for these manned systems [19].

1.3 Contributions

The three questions stated at the beginning of this chapter regard the coordination problem from a modeling, algorithmic and application point of view. This section will review the contributions in this thesis, starting with the modeling parts, which are the most numerous.

The starting point of this thesis is an extension of the approach in [12], in which the system dynamics were abstracted by parameterizing the energy consumption of each robot task as a function of its execution time. The parameterization in [12] was based on a time scaling of existing trajectories. Our extended approach instead embeds the energy minimal solutions into the parameterized energy functions. It is shown how these improved functions may be generated by posing a space discretized model, and solving the problem using dynamic programming. Only a single dynamic programming problem needs to be solved, since its state space grid will cover the range of relevant execution times.

A drawback with the parameterization approach is that robots must reach a stand still in between tasks. This makes a tight synchronization difficult, e.g. if one robot follows behind another. Rather than using parameterization as earlier, additional detail could be encoded into the mathematical program. In an effort to add more general detail, we regard synchronized hybrid systems by modeling the continuous states using collocation, and communication between systems using the scheduling style constraints from our previous work. We present a numerical example where the resulting formulation is applied to dual double integrator systems. Part I of this thesis also includes an extended discussion on possible numerical issues related to the synchronization.

While the approach successfully adds more detail to the problem formulation, it is also computationally demanding due to its nonlinearities. Next, we propose two monolithic minimum time/energy models, specifically for the coordination problem. The models utilize both space discretization and variable transformations to remove difficult nonlinearities. Furthermore, by using path based criteria in combination with an inner approximation of acceleration constraints, one of the models is reduced to a linearly constrained quadratic program.

The final modeling contribution concerns a slightly different problem, where three stacker cranes all sharing a single track are to be scheduled. The objective is to minimize the final time, and the decisions include task assignment, sequencing

and trajectory generation. Building on an existing approach, we pose a model for the stacker crane problem. Furthermore, we introduce a model abstraction, which decreases the problem complexity. Part I of this thesis also extends the formulation to bounded acceleration.

On the algorithmic side, we have focused on the use of constraint propagation methods to speed up the branch and bound phase required in the coordination problem. In a conference paper [20], we integrated constraint programming with state of the art mathematical programming methods. And in a case study, we demonstrated an order of magnitude speed up for a problem instance of the parameterization approach. At the time, the integration of constraint propagation and linear programming had been used to achieve impressive speedups for many problems, but the nonlinear case had not been investigated as closely. Today, several software make use of constraint propagation also for nonlinear programming. The appended papers include an extended version of [20], which is focused on the use of constraint propagation for nonlinear scheduling problems. Based on the parameterization and monolithic approach, we define two distinct problem classes, and in a thorough benchmark, we compare existing methods to our proposed formulation.

As for quantification of energy reduction, we employ accurate energy measurement equipment and outline a reliable methodology to produce replicable experiments. Using a single robot, we compare four different cost functions and evaluate the effect of different robot payloads. Also peak-power consumption is examined. Two case studies for the two robot case are presented.

1.4 Thesis outline

The thesis is outlined as follows. In Chapter 2, the coordination problem is formalized, and we take a closer look at the collision avoidance constraints. The chapter is concluded by a discussion on the hierarchical properties of the problem and a review of algorithms relevant to this thesis.

As we will find out in Chapter 2, the collision avoidance problem gives rise to binary decisions which must be explored by branch and bound, splitting our problem into a high level sequencing problem, and a lower level continuous subproblem. Chapter 3 is focused on modeling the latter and discusses two approaches to the problem: a monolithic approach which models the subproblem using quadratic and nonlinear programming; a decomposition approach in which parts of the problem are parameterized. Chapter 4 addresses the sequencing problem and how traditional mathematical programming can be improved with constraint propagation methods. The primary application for our methods is that of energy minimal control of robotic manipulators. Chapter 5 presents experimental results and review factors impacting the energy consumption of the systems.

Next, we broaden the problem formulation somewhat. In Chapter 6 we discuss the relation of our methods to hybrid systems, and in Chapter 7 the stacker crane scheduling problem is investigated. Finally, Chapter 8 shortly summarizes each of the appended papers, and Chapter 9 concludes the thesis. The latter chapter also includes a discussion on possible future work.

Chapter 2

Problem formulation

Consider n systems, each defined by a normalized path position function $s_i : [0, T] \to [0, 1]$, $i \in \{1, \ldots, n\} = \mathcal{N}$, with $s_i(0) = 0$, $s_i(T) = 1$, $\dot{s}_i \geq 0$ where $\dot{(\cdot)} = d/dt$ and $T \in \mathbb{R}^+$ is the final time. Also suppose each system follows a fixed path $\mathbf{f}_i : [0, 1] \to \mathbb{R}^{m_i}$, where m_i is the dimensionality of the i:th system's position space. Each system's spatial position vector $\mathbf{q}_i(t)$ along its path $\mathbf{f}_i$ is defined as a function of the path position

$$\mathbf{q}_i(t) = \mathbf{f}_i(s_i(t)). \tag{2.1}$$

It follows from differentiation that the time derivatives of $\mathbf{q}_i$ are

$$\dot{\mathbf{q}}_i(t) = \mathbf{f}_i'(s_i)\dot{s}_i(t), \tag{2.2}$$
$$\ddot{\mathbf{q}}_i(t) = \mathbf{f}_i'(s_i)\ddot{s}_i(t) + \mathbf{f}_i''(s_i)\dot{s}_i(t)^2, \tag{2.3}$$

where $(')$ and $('')$ denote the first and second positional derivatives. Note that in the special case that a path does not have curvature, $|\mathbf{f}_i''|_\infty = 0$, the resulting problem becomes much simpler. These equations have been used for single robotic manipulator motion planning since at least the early 80'ies [21].

As stated in our scope, detailed model parameters are considered unknown. As such, bounding the velocity and acceleration will suffice. The bounds become

$$\mathbf{v}_i^{\min}(s_i) \leq \dot{\mathbf{q}}_i(t) \leq \mathbf{v}_i^{\max}(s_i), \tag{2.4}$$
$$\mathbf{a}_i^{\min}(s_i) \leq \ddot{\mathbf{q}}_i(t) \leq \mathbf{a}_i^{\max}(s_i), \tag{2.5}$$

where $\mathbf{v}_i^{\min}$ and $\mathbf{v}_i^{\max}$ are bounds on the velocity, and $\mathbf{a}_i^{\min}$ and $\mathbf{a}_i^{\max}$ are bounds on the acceleration.

It is however not enough to find a feasible solution, we would also like to minimize a weighted criterion on the form

$$w_1 T + w_2 \sum_{i=1}^{n} \int_0^T c_i \left(s_i(t), \dot{s}_i(t), \ddot{s}_i(t)\right) dt, \tag{2.6}$$

where T, as before, is the final time, $c_i : \mathbb{R}^3 \to \mathbb{R}$, $i \in \mathcal{N}$, is a system dependent criterion, while w_1 and w_2 are constant weights, penalizing the final time and the integrand. In this work, each criterion c_i should be correlated with the energy consumption of its system. The choice of criterion is very much dependent on the choice of model, and will be discussed in Chapter 3, as the models are presented.

Finally, the systems are subject to a collision avoidance constraint

$$\mathbf{s}(t) \notin \mathcal{F} \quad \forall t \in [0, T], \tag{2.7}$$

where $\mathbf{s} = [s_i]$ is the vector of path functions and $\mathcal{F}$ is a collision set. The underlying structure of this collision set will be discussed in the following subsection. Within robotics, the concept of collision sets have been used for more than 25 years, beginning with the two robot problem studied in [22]. Although at that time, and in later work on coordination of multiple robots [23], [24], none of the dynamics we introduced earlier were considered.

From an application perspective, the collision avoidance constraint shares many similarities to the problems found in intelligent transportation systems. Linear criteria include the average and maximum vehicle delay [17], which are similar to due dates and cycle time in robotics and scheduling. Much like minimum energy or jerk formulations for robots, there are dynamic criteria such as velocity tracking and actuator/discomfort penalties for transportation systems, e.g. as in [18]. There is naturally a somewhat greater emphasis on guaranteeing safety for these manned systems [19].

This chapter is specifically focused on the problem formulation, so we will leave the discussion on solution methods for Chapter 3. Now, let us summarize the problem formulation: the velocity and acceleration are described by (2.2)-(2.3), and then bounded by (2.4)-(2.5). The criterion (2.6) provides a means for selecting a solution and the collision avoidance constraint (2.7) describes the coupling between systems. In the next section, we will take a closer look at the collision avoidance constraint, whereafter we discuss the overall structure of the problem. The chapter is concluded with a review of algorithms relevant to the work in this paper.

2.1 Collision avoidance

The collision set $\mathcal{F}$ is not just any set. Because it describes collisions, it has a special structure. This section will delve deeper into that structure and pose (2.7) on a form suitable for mathematical programming. Throughout this section we will use an example to visualize our results. Consider the following scenario illustrated in Figure 2.1. Three robots share a common zone located in the interval $s_i \in [0.3, 0.7]$, and a collision occurs whenever two or more of the robots occupy the interval. This example is identical to what an intersection in a general multi-agent collision avoidance problem would look like [25].

Collisions are a pairwise phenomenon, and the collision set may thus be described as the union of several smaller two-dimensional sets describing the pairwise interaction between systems

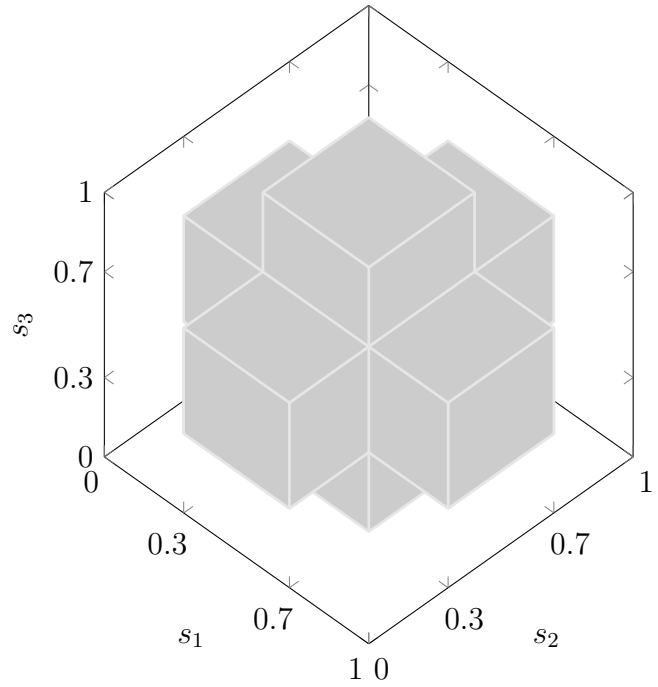

Figure 2.1: The coordination diagram for three robots with a common zone located in the interval $s_i \in [0.3, 0.7]$, $i \in [1, 2, 3]$, i.e. at most one robot may reside in $[0.3, 0.7]$ at any time.

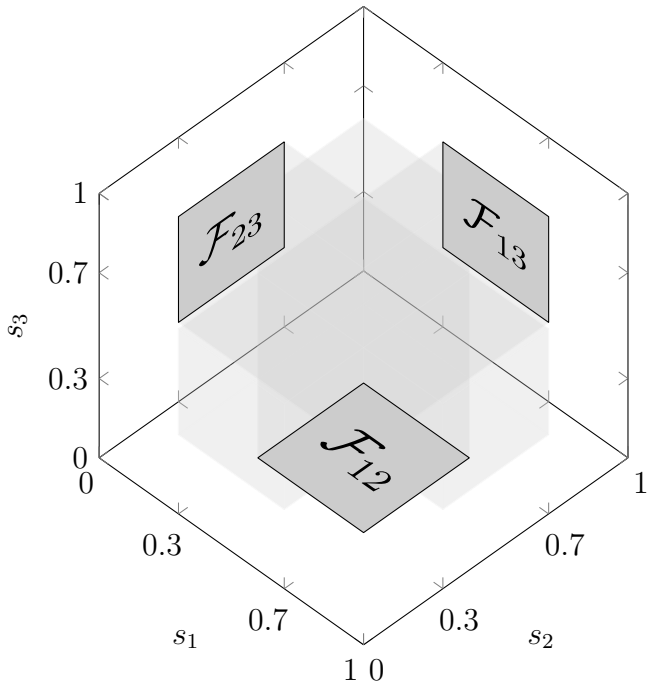

Figure 2.2: Three two-dimensional collision sets used to construct the coordination diagram in Figure 2.1.

$$\mathcal{F} = \bigcup_{\substack{i,j \in \mathcal{N}: \\ i<j}} \{\mathbf{s} : \langle s_i, s_j \rangle \in \mathcal{F}_{ij}\}, \tag{2.8}$$

where $\mathcal{F}_{ij} \subset [0,1] \times [0,1]$ are such two-dimensional sets, defining the collisions for pairs of systems $\langle i, j \rangle$. Figure 2.2 illustrates this result applied to our example. One can regard the complete collision set as composed of several two-dimensional sets projected into the other dimensions.

Instead of posing the collision avoidance constraint (2.7), these smaller two-dimensional sets may be used to pose the equivalent condition

$$\langle s_i(t), s_j(t) \rangle \notin \mathcal{F}_{ij} \qquad \forall t \in [0,T] \;\; i,j \in \mathcal{N} : \;\; i < j. \tag{2.9}$$

From here on we shall use this pairwise form. It should be noted that as we have decomposed the collisions into pairwise sets some information is no longer explicitly available to us, something that will be discussed in Chapter 4. Also, in this particular example there is only a single collision zone. In a more general setting this might not be the case. However, the more general case can be reduced to the single collision form by simple partitioning of the collision zones into individual sets. Hence, we will continue using only a single collision zone per set, without loss of generality.

Collision zones naturally give rise to disjoint solution spaces, i.e. a binary alternative where one system must be given priority over the other. Because of this, we need to model our problem such that our algorithms explore both regions. This is done by explicitly enumerating the alternatives, i.e. one robot should enter the zone before the other.

In other words, given a pairwise collision zone $\mathcal{F}_{ij}$, we can build on (2.9) and pose the collision avoidance constraint (2.7) as a disjunction between inequalities.

$$\begin{aligned} s_j(t) \geq \max\{s_b : \{s_a, s_b\} \in \mathcal{F}_{ij} : s_a \leq s_i(t)\} \quad & \forall t \in [0,T] \;\; \vee \\ s_i(t) \geq \max\{s_a : \{s_a, s_b\} \in \mathcal{F}_{ij} : s_b \leq s_j(t)\} \quad & \forall t \in [0,T] \\ & \forall i,j \in \mathcal{N} : \;\; i < j, \end{aligned} \tag{2.10}$$

where $s_a \leq s_i(t)$ and $s_b \leq s_j(t)$ are inequalities rather than equalities, in order to embed the fact that the systems position states are monotonically increasing. Note that to make these equations to be well defined over $[0,1]$, we will simply assume that the max function over the empty set is 0. For notational simplicity we use the following expression

$$\begin{aligned} s_j(t) \geq g_{ij}(s_i(t)) \quad & \forall t \in [0,T] \;\; \vee \\ s_i(t) \geq g_{ji}(s_j(t)) \quad & \forall t \in [0,T] \\ & \forall i,j \in \mathcal{N} : \;\; i < j, \end{aligned} \tag{2.11}$$

where $g_{ij} : \mathbb{R} \to \mathbb{R}$ and $g_{ji} : \mathbb{R} \to \mathbb{R}$ are the two max expressions from (2.10). Figure 2.3 illustrates the resulting functions for the pairwise collision set $\mathcal{F}_{12}$ from our example.

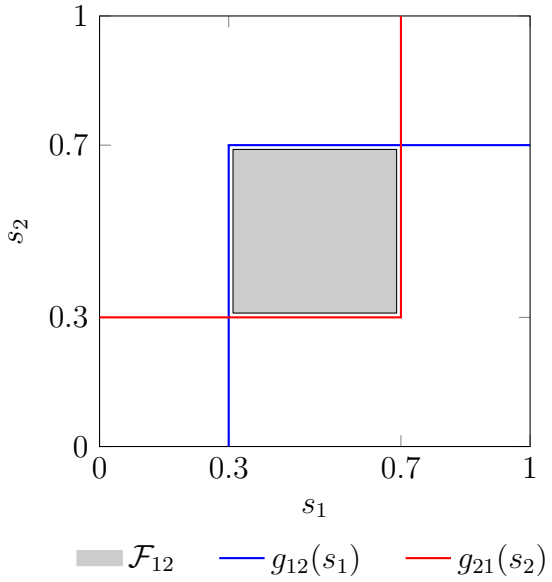

Figure 2.3: The collision set $\mathcal{F}_{12}$ and its two boundary functions $g_{12}(s_1)$ and $g_{21}(s_2)$ which may be used to pose our collision avoidance constraint.

Finally, instead of expressing our collision avoidance constraint as a function of the path position, we transform it into constraints on the system time as a function of its position. This is necessary for a state space discretization approach later employed. Let $t_i(s_i)$ be the time at which system i visits position s_i, and $t_j(s_j)$ the equivalent for system j. The timing conditions equivalent to (2.10), and in turn (2.7), is

$$\begin{aligned} t_j(g_{ij}(s_i)) \leq t_i(s_i) \quad & \forall s_i \in [0,1] \ \vee \\ t_i(g_{ji}(s_j)) \leq t_j(s_j) \quad & \forall s_j \in [0,1] \\ & \forall i,j \in \mathcal{N} : \ \ i < j. \end{aligned} \tag{2.12}$$

The first line can be thought of as: system i cannot visit s_i until system j has visited $g_{ij}(s_i)$.

To conclude this subsection, we will discuss how discretization affects the coordination diagram. The collision avoidance constraint takes the form of timing conditions, and in some of our approaches, it is beneficial to keep the number of these constraints to a minimum, i.e. keep the resolution of the discretization to a minimum. Thus, the constraint should only be applied when g_{ij} or g_{ji} varies. In the example used so far, g_{ij} or g_{ji} each only vary at one point, and it is enough to pose

$$t_j(0.7) \leq t_i(0.3) \vee t_j(0.3) \geq t_i(0.7). \tag{2.13}$$

It seems that for intersections, only a very low resolution is required for the timing conditions. Also note that this constraint looks very much like the finite resource conditions found in traditional scheduling [26].

There are however other situations where a higher resolution is required. Let us examine the case of two 1-DoF robot arms, an example which will be used throughout

this thesis, and for which we can derive an analytical expression. Suppose the two arms i and j are of length ℓ_i and ℓ_j, and positioned at a coordinate $\langle p_i^x, p_i^y \rangle$ in 2d-space.

Consider a single 1-DoF robot arm and the line extending from its base along the arm, the 2d coordinates along that line is defined by

$$\begin{bmatrix} p_i^x + d_i \cos(\theta_i) \\ p_i^y + d_i \sin(\theta_i) \end{bmatrix}, \tag{2.14}$$

where d_i is the position along robot i's line. Now for each pair of robots i and j, the intersection point for the robots' lines is

$$\begin{bmatrix} p_i^x + d_i \cos(\theta_i) \\ p_i^y + d_i \sin(\theta_i) \end{bmatrix} = \begin{bmatrix} p_j^x + d_j \cos(\theta_j) \\ p_j^y + d_j \sin(\theta_j) \end{bmatrix}, \tag{2.15}$$

which by solving for d_i and d_j yields

$$\begin{bmatrix} d_i \\ d_j \end{bmatrix} = \frac{1}{\sin(\theta_i - \theta_j)} \begin{bmatrix} \sin(\theta_j) & -\cos(\theta_j) \\ \sin(\theta_i) & -\cos(\theta_i) \end{bmatrix} \begin{bmatrix} p_i^x - p_j^x \\ p_i^y - p_j^y \end{bmatrix}. \tag{2.16}$$

A collision occurs when both $0 \leq d_i \leq \ell_i$ and $0 \leq d_j \leq \ell_j$. If we translate this to the interval $[-1, 1]$, collisions occur when

$$\left\| \begin{matrix} 2d_i/\ell_i - 1 \\ 2d_j/\ell_j - 1 \end{matrix} \right\|_\infty \leq 1. \tag{2.17}$$

Figure 2.4 illustrates the resulting collision zone of two robots with unit length arms at distance $\sqrt{2}$ along the x-axis, moving from a top to a bottom pointing position. Using the boundary of the collision zone, we can derive the two collision functions g_{ij} or g_{ji} which are illustrated in blue and red lines respectively. Clearly, a higher resolution is warranted, compared to that of the intersection case, unless of course the two robots were to follow each other through the intersection.

In this particular case, it was possible to derive an analytical expression for the collision set. A more practical approach for more complex paths is to create a sweep volume for each robot, and then compute the intersection between sweep volumes to identify where collisions occur. This is an approach which has been used for both car-like [27] and industrial [28] robots.

2.2 Structure

Let us recall our problem definition: construct n trajectories with bounds defined by (2.2)-(2.5), which minimize the criteria (2.6), while avoiding collisions (2.7). In the previous section we learnt that the collision avoidance condition can be transformed

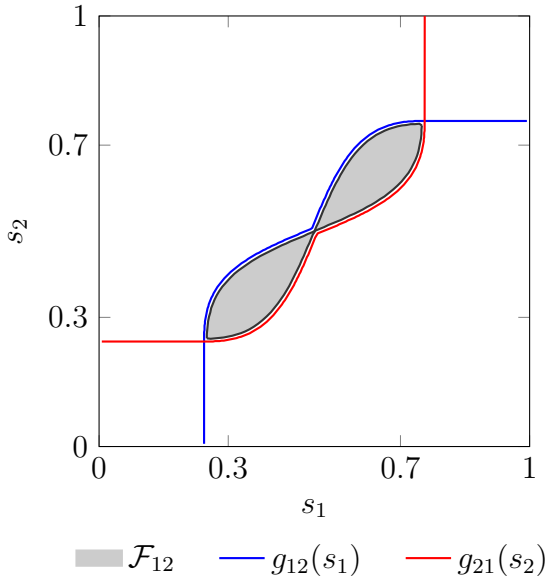

Figure 2.4: The collision set $\mathcal{F}_{12}$ for the 1-DoF robot case, and its two boundary functions $g_{12}(s_1)$ and $g_{21}(s_2)$.

in to disjunct timing conditions. Furthermore, unless the systems are in fact following each other, it may not be necessary to impose these timing conditions at all points along the trajectory. This section will briefly discuss the hierarchical structure that appears when the timing conditions only need to be posed at few points.

First, regardless of the level of discretization, the top most level of the problem consists of the binary decisions imposed by the disjunct timing conditions. These must be explored using branch and bound. Note that the mathematical programing relaxation for such a disjunct timing condition is simply ignoring the condition entirely, which makes the relaxation very weak. This weakness is a well known problem which occurs in scheduling models, and causes search algorithms to delve deeply into infeasible branches before detecting infeasability [29].

Regarding the discretization, consider the collision set in Figure 2.4 resulting from the two 1-DoF robot example in the previous section. Let us focus in particular on the case where the second robot has priority, i.e. the region above g_{12} (blue line). To model this region, the first most term in the collision avoidance constraint (2.12) should hold

$$t_2(g_{12}(s_1)) \leq t_1(s_1) \quad s_1 \in [0.25, 0.75]. \tag{2.18}$$

Note that because of the monotonicity property, it is enough to specify the interval $s_1 \in [0.25, 0.75]$, as the constraint will implicitly hold outside.

One could regard the two systems as being independent in the interval $[0, 0.25]$, next completely coupled during $[0.25, 0.75]$ and finally two independent systems once again for $[0.75, 1]$. To decrease the coupling of the systems, it is possible to work with an approximation of $g^a_{12}(s_1)$. Figure 2.5 illustrates such an approximation with

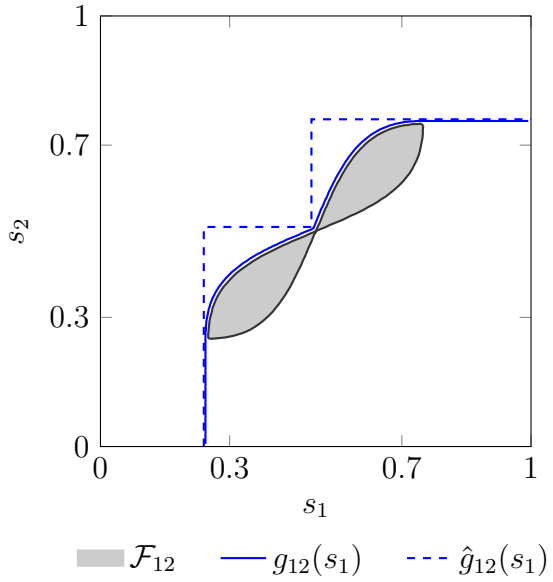

Figure 2.5: The collision set $\mathcal{F}_{12}$ for the 1-DoF robot case, the boundary function $g_{12}(s_1)$ and an approximation $\hat{g}_{12}(s_1)$.

a dashed blue line. Using this new approximation, it is enough to pose the collision avoidance constraints at coordinates $\langle 0.25, 0.5\rangle$ and $\langle 0.5, 0.75\rangle$ in s_1/s_2-space, i.e.

$$t_2(0.5) \leq t_1(0.25), \tag{2.19}$$
$$t_2(0.75) \leq t_1(0.5). \tag{2.20}$$

Using this approximation the systems can be considered independent for all but the two synchronization points. The level of synchronization for the exact and approximate case is illustrated in Figure 2.6.

We note that the approximate case in Figure 2.6 looks much like two multi stage systems, each with two switching points. But rather than some change of dynamics or discrete state jump, the time of transition is coupled between systems. We will discuss this perspective, as well as the applications of our method to these more general systems, in Chapter 6. In the following chapter, we will focus on the subproblem where the binary sequence decisions have already been decided, and in the following chapter, we will address the algorithmic problem of weak relaxations.

2.3 Algorithms

In Chapter 4, the robot coordination problem will be posed as a mixed integer nonlinear program (MINLP), and constraint programming is used to improve the performance of existing algorithms. This section will provide some necessary background on MINLP algorithms and constraint programming. Although quadratic programming and pure nonlinear programming are also used in this thesis, we do not present any algorithmic improvement for these, and thus do not discuss the underlying algorithms.

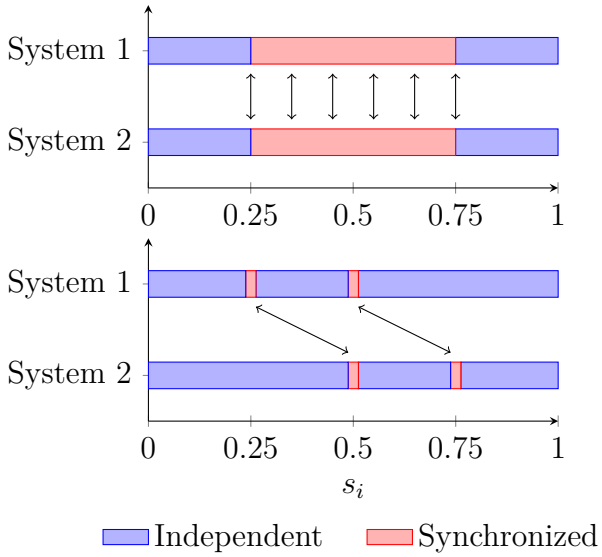

Figure 2.6: The level of synchronization between two systems using the boundary function $g_{12}(s_1)$ (top) and its approximation $\hat{g}_{12}(s_1)$ (bottom).

2.3.1 Mixed integer nonlinear programming

There are a number of different MINLP solution methods, each implemented in variety of software packages [30]. For non-convex MINLPs, i.e. when the integer relaxed problem is non-convex, nonlinear programming based branch and bound (NLP-BB) is typically used [31], [32]. If the problem is convex, which this thesis is focused on, there are additional choices, e.g. outer approximation [33], [34], linear and nonlinear programming based branch and bound (LP/NLP-BB) [35] and extended cutting plane [36]. All these algorithms use some type of branch and bound, a general search method, which systematically divides (branches) the feasible space and attempts to identify all candidate solutions. Large portions of the feasible space may be discarded based on estimated lower and upper bounds (bounding) on the optimization criteria.

If the problem is convex or if locally optimal solutions are sufficient, NLP-BB works as follows. A tree of problems yet to be processed is defined. The top level node in this tree is initialized with an integer relaxation of the MINLP problem, resulting in a nonlinear program (NLP). The solution to this relaxed problem provides a lower bound on the objective value for the lower branches. Iteratively, integer variables are branched upon and constrained, such that NLP subproblems, with some integers fixed and other relaxed, are created. The resulting solution of each NLP subproblem yields a lower bound for the current branch. When all integer variables have been fixed such that they form a complete integer assignment, the solution provides an upper bound for the entire MINLP. Once an upper bound has been found, any nodes on the tree with a greater lower bound than the current upper bound can be removed from the tree of problems yet to be processed. The algorithm terminates when all branches of the tree have been explored or removed.

The LP/NLP-BB algorithm was first presented in [35]. In short, LP/NLP-BB just as outer approximation, utilizes the fact that solving a MINLP is equivalent of

Algorithm 1 NLP/LP-Based Branch-and-Bound Algorithm

1: **Initialize.**
Populate the search tree Γ with the root node, initialize the linear programming MP, set the upper bound $c_{ub} = c_{max}$.
2: **Terminate?**
If the search tree is empty, $\Gamma = \emptyset$, terminate.
3: **Select.**
Select the next node γ from Γ.
4: **Evaluate.**
Solve linear MP based on the full/partial integer assignment p, if infeasible goto 2.
5: **Prune.**
If the current solution $c_{MP}(\gamma) \geq c_{ub}$, goto 2.
6: **Solve NLP?**
If we are at a leaf node (full integer assignment), solve the NLP corresponding to γ, else goto 9.
7: **Upper bound?**
If the NLP solution is feasible and cost $c_{NLP}(\gamma) \leq c_{ub}$, update the upper bound.
8: **Refine.**
Add linear approximations to the MP based on the NLP solution, goto 2.
9: **Divide.**
Branch on current node, add the new nodes to the search tree Γ, goto 2.

solving a mixed integer linear program of finite size [34]. Suppose we could identify all feasible integer solutions of a problem, and for each integer solution, linearize the problem at the locally optimal solution. Solving the resulting linear approximation would then yield the same solution as solving the MINLP.

Of course, such a large number of linearizations is intractable, and in the process of finding all the linearization points would have already found the optimal solution. Instead, the algorithm will iteratively build the approximation, typically denoted reduced master problem (MP). The overall execution of the algorithm is described in Algorithm 1, adapted from [30]. In step 1, an initial linearization is created at the solution of the NLP resulting from the integer relaxed MINLP. We will use c to denote the complete objective function cost, c_{MP} the master problem solution, c_{ub} the current upper bound and c_{NLP} the local NLP solution. Steps 2-3 and 9 provides the base functionality of the branching search. At each node, a linear approximation is solved (step 4). If it is feasible and the solution is not bounded by the current upper bound (step 5), and we are at a leaf node, an NLP is solved (step $6-7$). Finally, additional linearizations may be added to the MP (step 8). Different rule sets exist which define how often an NLP should be generated. Preferably, there should also exist a mechanism for the removal of unused linearizations.

One drawback of MINLP methods for nonlinear scheduling problems is the weak lower bound resulting from the integer relaxation. In a mutual exclusion constraint where for example two time intervals may not overlap, such as the collision avoidance

constraints in our problem, the relaxation of the constraint will label the entire search space feasible. As such, infeasibilities which are caused by discrete dynamics or logic are not detected until far down in the search tree. There are methods, such as convex hull relaxations [37], which by adding auxiliary variables and constraints aim to increase the tightness of the relaxations. Although showing promising results for a variety of problems, for scheduling disjunctions, a convex hull formulation is of little or no help [29].

2.3.2 Constraint programming

Constraint programming is a field with roots in the artificial intelligence and computer science communities. Originally used for solving constraint satisfaction problems, i.e. assigning values to variable such as to satisfy a set of constraints, constraint programming has evolved during the last decade to also solve optimization problems. For an introduction to constraint programming we refer to [38], [39]. In contrast to classic mathematical programming methods, constraint programming offers a very rich modeling language, where constraints are not limited to equalities and inequalities. For example, in a scheduling application, one may use a single specialized constraint to describe mutual exclusion.

Constraint programming uses a branch and bound search similar to that of mathematical programming methods. However, instead of solving relaxations at each node, propagation methods are used to infer information about the domain of each decision variable. Propagation can be regarded as follows: the set of possible values for each variable is kept track of, and is reduced based on the problem constraints. Each constraint has an associated propagation method which is scheduled for execution by the search method. A propagator may be executed more than once in each node if deemed necessary. Typically, a propagator subscribes to a number of variables (those included in the constraint) and is scheduled whenever the domain of those variables change. After a propagator has been executed, it either enters a state where it can be rescheduled, or one where it wont.

The following example will demonstrate a simple propagator. Consider two variables $x, y \in \{1, 2, 3, 4, 5\}$ subject to the inequality $x+2y \leq 5$. A propagator for this constraint can reduce the upper bound on the variables by applying $x_{max} := 5-2y_{min}$ and $y_{max} := (5 - x_{min})/2$. The first will reduce the domain of x to $\{1, 2, 3\}$ and the second the domain of y to $\{1, 2\}$.

We mentioned previously that there exist specialized propagators for mutual exclusion. A basic disjunctive scheduling constraint [40], is defined as follows

$$\text{disjunctive}(\mathbf{t}, \mathbf{d}), \tag{2.21}$$

where $\mathbf{t} \in \mathbb{R}^k$ is a vector of k operation start times and $\mathbf{d} \in \mathbb{R}^k$ is a vector of corresponding durations, all utilizing the same resource. The constraint enforces that for any pair of operations, one must finish before the other. This is known as a unary constraint in Gecode [41] and a nooverlap constraint in IBM ILOG CP Optimizer. Propagation methods for this mutual exclusion constraint may be based on edge finding [42].

Furthermore, in contrast to branch and bound methods in mathematical programming, branching in constraint programming does not necessarily imply the branching on the domain of a variable. It could just as well apply constraints to divide the feasible solution set. One may regard constraint programming as taking a local approach to the problem, examining each constraint by itself, along with its associated variables. Mathematical programming methods on the other hand, will take a global approach, regarding all variables and constraints at the same time. While constraint programming has proven successful in a wide variety of areas, e.g. assignment problems, scheduling, transport, personnel planning, etc, it does have its drawbacks. Variables are most often limited to integers, and propagation for general nonlinear constraints can be considered weak at best. Thus constraint programming is currently intractable to use for a large number of applications, e.g. discretized continuous state variables in optimal control problems.

2.4 Summary

To summarize, the problem formulation is defined by (2.2)-(2.7). These include system dynamics in the form of bounded velocity and acceleration along a fixed path, criteria to be minimized, and a collision avoidance constraint. In Section 2.1, the collision avoidance constraint (2.7) was rewritten as the disjunctive timing constraint (2.12). An analytic instance of (2.12) was posed for a rotating robot arm example, for later use in Chapters 4 and 6.

The underlying structure of the problem was discussed in Section 2.2, where most importantly, the problem may be decomposed into a sequencing problem and a continuous subproblem. The former entails deciding in which temporal order robots move through shared zones, and the latter involves computing the continuous robot trajectories when the sequence is known. Chapters 4 and 3 will discuss these two problems, respectively. Finally, relevant mathematical and constraint programming concepts were reviewed, these are relevant for the algorithmic improvements in Chapter 4.

Chapter 3

Fixed sequence subproblem

The robot coordination problem entails both deciding the sequence in which robots move through the shared zones, as well as the velocity profile of each individual robot. A branch and bound search is typically applied to the sequencing problem. During this search, full and partial sequences are evaluated. This chapter is focused on that subproblem, where the sequence is known and the velocity profiles are to be decided. We present two approaches. A monolithic approach, suitable when the collision set has high resolution characteristics, and a decomposition approach, more suited for low resolution collision sets.

3.1 Monolithic approach

The single robot trajectory planning problem is often modeled by considering the velocity *along* the path (path velocity) as a state variable, rather than the velocity of the actual robot joints [21], [43]. While this reduces the dimensionality of the problem greatly, it also introduces nonlinearity in the path function that translates path position into joint positions. Furthermore, the addition of collision avoidance introduces binary variables which describe the priority sequence of crossing the shared zone, as well as the occupancy state that prevents two robots to reside in the shared zone at the same time. The problem is typically solved using nonlinear programming [9], [10], [43] or dynamic programming [21], [44]. For the multi robot case, dynamic programming would be intractable due to the curse of dimensionality. Therefore a mathematical programming approach is needed.

This section will present two monolithic models for the problem. The first makes use of a convexity result for single robots found in [43]. Unfortunately the problem does not remain convex with the addition of timing constraints found in the multi robot case. But even so, the model may still be of use. The second model is based on another convexification result found in [18], where it is used for traffic intersections. We add joint velocity and acceleration constraints, and using linearization, the second model becomes a linearly constraint quadratic program. We will be using velocity and acceleration as in (2.2)-(2.3), bounds as (2.4)-(2.5), and collision avoidance constraint (2.12). As for the criteria, we will discuss possible choices after the models have been introduced.

When the sequence is unknown, boolean variables are required to model the disjunctive collision avoidance constraint (2.12). But in this chapter, the sequence is assumed to be known and the collision avoidance constraints can simply be stated as

$$t_j(g(s_i)) \leq t_i(s_i) \quad \forall s_i \in [0,1] \;\; \forall \langle i,j,g(\cdot)\rangle \in \mathcal{G}, \tag{3.1}$$

where $\mathcal{G}$ is a set of tuples $\langle i,j,g(\cdot)\rangle$, where each tuple describes robots i and j having a collision function $g(\cdot)$. That is, robot i cannot visit location s_i before robot j has been at position $g(s_i)$. A general time formulation can be summarized as

$$\begin{aligned}
&\min \left[w_1 \max_{i\in\mathcal{N}} (T_i) + w_2 \sum_{i\in\mathcal{N}} \int_0^{T_i} c_i(s_i(t_i))dt_i \right] \\
&\text{subject to} \\
&\left.\begin{array}{ll} s_i(0) = 0 & s_i(T_i) = 1 \\ \dot{s}_i(0) = v_i^0 & \dot{s}_i(T_i) = v_i^f \end{array}\right\} \qquad \forall i \in \mathcal{N}, \\
&\left.\begin{array}{l} |\mathbf{f}_i'(s_i(t))\dot{s}_i(t)| \leq \mathbf{v}_i^{\lim}(s_i(t)) \\ |\mathbf{f}_i'(s_i(t))\ddot{s}_i(t) + \mathbf{f}_i''(s_i(t))\dot{s}_i(t)^2| \leq \mathbf{a}_i^{\lim}(s_i(t_i)) \end{array}\right\} \\
&\qquad\qquad \forall t \in [0,T_i] \;\; i \in \mathcal{N}, \\
&t_j(g(s_i)) \leq t_i(s_i) \quad \forall s_i \in [0,1] \;\; \forall \langle i,j,g(\cdot)\rangle \in \mathcal{G},
\end{aligned} \tag{3.2}$$

where both s_i and t_i are optimization variables, and T_i could be either a variable or a constant, v_i^0 and v_i^f are bounds on the initial and final velocity of system i. The velocity and acceleration bounds have been made symmetric without loss of generality, i.e. $\mathbf{v}_i^{\lim} = \mathbf{v}_i^{\max} = -\mathbf{v}_i^{\min}$ and $\mathbf{a}_i^{\lim} = \mathbf{a}_i^{\max} = -\mathbf{a}_i^{\min}$.

3.1.1 Space formulation

In order to reduce nonlinearity, the trajectory planning problem (in the case of single robots) is often solved by space discretization rather than time [21], [43], [44]. In the mathematical programming case, a space formulation is advantageous because it reduces the nonlinearity of the path function [43]. The path function can now be regarded as a parameter variation in the space domain, rather than a nonlinearity in the position state.

The model (3.2) is a hybrid model with expressions both in time and space, and somehow, these elements must be bridged. In a time formulation, binary variables would be used to express the occupancy of a zone over time. In a space formulation on the other hand, no such binary variables are needed. Instead, we must transform the dynamics and bounds to the space domain.

When the problem is posed in space, it entails determining the path velocity and acceleration as a function of the *path position* for each system, that is $\dot{s}_i(s_i)$ and $\ddot{s}_i(s_i)$ for $s_i \in [0,1]$. Furthermore, the variable $t_i(s_i)$ which defines the time at which system i is at position s_i is now treated as a system state. The position s_i on the other hand now acts as a spatial dimension for the state space rather than a system state.

We note that the relationship between time and path velocity becomes

$$t_i'(s_i) = \frac{dt_i(s_i)}{ds_i} = \frac{1}{\dot{s}_i(t_i(s_i))} = \frac{1}{\dot{s}_i(s_i)}, \tag{3.3}$$

or on integral form

$$t_i(s_i) = \int_0^{s_i} \frac{1}{\dot{s}_i(s_i)} ds_i. \tag{3.4}$$

The boundary conditions on position are replaced with boundary conditions on time

$$t_i(0) = 0 \quad t_i(1) = T_i, \tag{3.5}$$

as for the boundaries of $\dot{s}_i(s_i)$, these become

$$\dot{s}_i(0) = v_i^0 \quad \dot{s}_i(1) = v_i^f. \tag{3.6}$$

The time dependence of the velocity and acceleration constraints are removed as these are enforced on the interval $s_i \in [0, 1]$. The timing conditions remain unchanged. Since the integral of the cost function is now in the space domain, the cost function becomes

$$w_1 \max_{i \in \mathcal{N}} (T_i) + w_2 \sum_{i \in \mathcal{N}} \int_0^1 \frac{c_i(s_i)}{\dot{s}_i(s_i)} ds_i. \tag{3.7}$$

The space formulation can be summarized as

$$\begin{aligned}
&\min \left[w_1 \max_{i \in \mathcal{N}} (T_i) + w_2 \sum_{i \in \mathcal{N}} \int_0^1 \frac{c_i(s_i)}{\dot{s}_i(s_i)} ds_i \right] \\
&\text{subject to} \\
&\left.\begin{array}{ll} t_i(0) = 0 & t_i(1) = T_i \\ \dot{s}_i(0) = v_i^0 & \dot{s}_i(1) = v_i^f \end{array}\right\} \qquad \forall i \in \mathcal{N}, \\
&\left.\begin{array}{l} t_i'(s_i) = 1/\dot{s}_i(s_i) \\ |\mathbf{f}_i'(s_i)\dot{s}_i(s_i)| \leq \mathbf{v}_i^{\lim}(s_i) \\ |\mathbf{f}_i'(s_i)\ddot{s}_i(s_i) + \mathbf{f}_i''(s_i)\dot{s}_i(s_i)^2| \leq \mathbf{a}_i^{\lim}(s_i) \end{array}\right\} \\
&\qquad\qquad \forall s_i \in [0, 1] \quad i \in \mathcal{N}, \\
&t_j(g(s_i)) \leq t_i(s_i) \quad \forall s_i \in [0, 1] \;\; \forall \langle i, j, g(\cdot) \rangle \in \mathcal{G},
\end{aligned} \tag{3.8}$$

where t_i, $\dot{s}_i$ and $\ddot{s}_i$ are optimization variables, and T_i, the final time of system i, is either a variable or a constant. Except for the slightly nonlinear square term $\dot{s}_i(s_i)$ in the acceleration equations, all the nonlinearities in the bounds have been transformed into parameter variations. However, the timing function used for final time and the collision avoidance constraints remains nonlinear.

Backward Euler model

A variation of this model is used in a conference version of Paper 4 [45], where the joint velocities and accelerations are explicitly modeled using a backwards difference approximation

$$\begin{cases} \dot{\mathbf{q}}_i(s_i) = \dfrac{\mathbf{q}_i(s_i) - \mathbf{q}_i(s_i - \Delta)}{t_i(s_i) - t_i(s_i - \Delta)}, \\ \ddot{\mathbf{q}}_i(s_i) = \dfrac{\dot{\mathbf{q}}_i(s_i) - \dot{\mathbf{q}}_i(s_i - \Delta)}{t_i(s_i) - t_i(s_i - \Delta)}, \end{cases} \tag{3.9}$$

where Δ is a fixed space sampling distance, $t_i(s_i)$, $\dot{q}(s_i)$ and $\ddot{q}(s_i)$ are variables and $q_i(s_i)$ is the fixed path. That is, the model does not consider the path velocity $\dot{s}_i$ as a state, but rather models all the individual joints explicitly using (3.9) and t_i as a control input. Due to the non-convexity of the model, an initial solution is required. This will be referred to as the explicit formulation.

3.1.2 Variable transformations

An additional strategy that further reduces nonlinearity is the introduction of variable changes. Two such variable changes have been explored in the literature: kinetic energy (squared velocity) [43] and inverse velocity [18]. Each removes a specific nonlinearity from the problem without introducing approximations. The former is used for single robot trajectory planning and linearizes the path function. Although not included here, additional torque equations can also be included in the first model. which would support applications such as space vehicles, aircraft and a large range of car models discussed in [46]. The latter is used for a vehicle intersection problem and linearizes the time-velocity used for the collision avoidance constraints.

Kinetic energy model

The kinetic energy formulation introduces a variable change where a state $e_i(t)$, proportional to the kinetic energy, is used instead of $\dot{s}_i$

$$e_i(s_i) = \dot{s}_i(s_i)^2, \tag{3.10}$$

which leads to the state equation

$$e_i'(s_i) = 2\ddot{s}_i(s_i). \tag{3.11}$$

where $e_i'(s_i)$ is the position derivative of $e_i(s_i)$. The relationship between time and path velocity becomes

$$t_i'(s_i) = \frac{1}{\dot{s}_i(s_i)} = \frac{1}{\sqrt{e_i(s_i)}}, \tag{3.12}$$

or on integral form

$$t_i(s_i) = \int_0^{s_i} \frac{1}{\dot{s}_i(s_i)} ds_i = \int_0^{s_i} \frac{1}{\sqrt{e_i(s_i)}} ds_i. \tag{3.13}$$

Using $e_i(s_i)$ and $e_i'(s_i)$ as the system states, the velocity and acceleration (2.2) and (2.3) are rewritten as

$$\dot{\mathbf{q}}_i(s_i)^2 = \mathbf{f}_i'(s_i)^2 e_i(s_i), \tag{3.14}$$
$$\ddot{\mathbf{q}}_i(s_i) \ = \mathbf{f}_i'(s_i) e_i'(s_i)/2 + \mathbf{f}_i''(s_i) e_i(s_i). \tag{3.15}$$

The velocity bound can be rewritten as

$$\dot{\mathbf{q}}_i(s_i)^2 \leq \mathbf{v}_i^{\lim}(s_i)^2, \tag{3.16}$$

and the the cost function is now

$$w_1 \max_{i \in \mathcal{N}} (T_i) + w_2 \int_0^1 \frac{c_i(s_i)}{\sqrt{e_i(s_i)}} ds_i. \tag{3.17}$$

In summary, the kinetic energy based formulation is

$$\begin{aligned}
&\min \left[w_1 \max_{i \in \mathcal{N}} (T_i) + w_2 \sum_{i \in \mathcal{N}} \int_0^1 \frac{c_i(s_i)}{\sqrt{e_i(s_i)}} ds_i \right] \\
&\text{subject to} \\
&\left.\begin{array}{ll} t_i(0) = 0 & t_i(1) = T_i \\ e_i(0) = \sqrt{v_i^0} & e_i(1) = \sqrt{v_i^f} \end{array}\right\} \qquad \forall i \in \mathcal{N}, \\
&\left.\begin{array}{l} t_i'(s_i) = 1/\sqrt{e_i(s_i)} \\ \mathbf{f}_i'(s_i)^2 e_i(s_i) \leq \mathbf{v}_i^{\lim}(s_i)^2 \\ |\mathbf{f}_i'(s_i) e_i'(s_i)/2 + \mathbf{f}_i''(s_i) e_i(s_i)| \leq \mathbf{a}_i^{\lim}(s_i) \end{array}\right\} \\
&\qquad\qquad \forall s_i \in [0,1] \quad i \in \mathcal{N}, \\
&t_j(g(s_i)) \leq t_i(s_i) \quad \forall s_i \in [0,1], \langle i, j, g(\cdot) \rangle \in \mathcal{G},
\end{aligned} \tag{3.18}$$

where t_i, e_i and e_i' are optimization variables, and T_i is either a variable or a constant. Most constraints are linear and the individual robot final time is convex. The differential equation (3.12) relating time to velocity is however nonlinear. While it would be possible to construct a linear inner approximation, it will not be useful in practice since the timing error will become intractable even for small to moderate changes in reference velocity. The non-convex term must be kept as is.

Inverse velocity model

The second transformation introduces the new state $z_i(s_i) = 1/\dot{s}_i(s_i)$, the inverse velocity (or lethargy). We can see from (3.13) that its spatial integral is time, and thus on differential form

$$t_i'(s_i) = z_i(s_i). \tag{3.19}$$

Also, introduce the spatial derivative of the inverse velocity $z_i'(s_i)$, which is related to $\ddot{s}(s_i)$ by

$$\ddot{s}_i(s_i) = -\frac{z_i'(s_i)}{z_i(s_i)^3}, \tag{3.20}$$

which follows from

$$\begin{aligned} \ddot{s}_i(s_i) &= \frac{d}{dt}\left[\frac{1}{z_i(s_i)}\right] \\ &= -\frac{1}{z_i(s_i)^2}\frac{dz_i(s_i)}{ds_i}\frac{ds_i}{dt}. \end{aligned} \tag{3.21}$$

Since time is now a state variable, the boundary conditions on time are simply linear state constraints

$$t_i(0) = 0 \quad t_i(1) = T_i. \tag{3.22}$$

The same goes for the boundaries of $\dot{s}_i$, that become

$$z_i(0) = \frac{1}{v_i^0} \quad z_i(1) = \frac{1}{v_i^f}. \tag{3.23}$$

The case of zero (or near zero) initial and final velocity will be discussed later.

The velocity (2.2) becomes

$$\dot{\mathbf{q}}_i(s_i) = \frac{\mathbf{f}_i'(s_i)}{z_i(s_i)}, \tag{3.24}$$

and since $z_i(s_i)$ is non-negative the velocity bound (2.4) can be expressed

$$|\mathbf{f}'(s_i)| \leq z_i(s_i)\mathbf{v}_i^{\lim}(s_i). \tag{3.25}$$

The acceleration (2.3) is

$$\ddot{\mathbf{q}}_i(s_i) = -\mathbf{f}_i'(s_i)\frac{z'(s_i)}{z_i(s_i)^3} + \mathbf{f}_i''(s_i)\frac{1}{z_i(s_i)^2}, \tag{3.26}$$

and the acceleration bound (2.5) consequently

$$|-\mathbf{f}_i'(s_i)z_i'(s_i) + \mathbf{f}_i''(s_i)z_i(s_i)| \leq z_i(s_i)^3\mathbf{a}_i^{\lim}(s_i). \tag{3.27}$$

The timing constraints (3.1) remain unchanged.

Finally for the criteria, integration in space adds an inverse velocity term just as in the kinetic energy model, such that

$$w_1 \max_{i\in\mathcal{N}}(T_i) + w_2 \sum_{i\in\mathcal{N}} \int_0^1 z_i(s_i)c_i(s_i)ds_i. \tag{3.28}$$

The inverse velocity formulation can be summarized as

$$\begin{aligned}
&\min \left[w_1 \max_{i\in\mathcal{N}}(T_i) + w_2 \sum_{i\in\mathcal{N}} \int_0^1 z_i(s_i)c_i(s_i)ds_i \right] \\
&\text{subject to} \\
&\left.\begin{array}{ll} t_i(0) = 0 & t_i(1) = T_i \\ z_i(0) = \frac{1}{v_i^0} & z_i(1) = \frac{1}{v_i^f} \end{array}\right\} \qquad \forall i \in \mathcal{N}, \\
&\left.\begin{array}{l} t_i'(s_i) = z_i(s_i) \\ |\mathbf{f}_i'(s_i)| \leq z_i(s_i)\mathbf{v}_i^{\lim}(s_i) \\ |-\mathbf{f}_i'(s_i)z'(s_i) + \mathbf{f}_i''(s_i)z_i(s_i)| \leq z_i(s_i)^3\mathbf{a}_i^{\lim}(s_i) \end{array}\right\} \\
&\qquad\qquad \forall s_i \in [0,1], i \in \mathcal{N}, \\
&t_j(g(s_i)) \leq t_i(s_i) \quad \forall s_i \in [0,1] \;\; \forall \langle i, j, g(\cdot)\rangle \in \mathcal{G},
\end{aligned} \tag{3.29}$$

where t_i, z_i and z_i' are optimization variables, and T_i is either a variable or a constant. Note that the only nonlinear constraint in this model is the acceleration bound. It is possible to make a linear inner approximation of the cubic term around a reference inverse velocity $\bar{z}_i(s_i)$ such that the acceleration bound becomes

$$\begin{aligned}
&|-\mathbf{f}_i'(s_i)z_i'(s_i) + \mathbf{f}_i''(s_i)z_i(s_i)| \leq \\
&\quad \left(\bar{z}_i(s_i)^3 + 3\bar{z}_i(s_i)^2\left(z_i(s_i) - \bar{z}_i(s_i)\right)\right)\mathbf{a}_i^{\lim}(s_i).
\end{aligned} \tag{3.30}$$

3.1.3 Criteria

Both the kinetic energy and the inverse velocity formulations have convex expressions of the final time. This section will focus on the minimum energy part of the criteria c_i. First, two convex criteria for the kinetic energy formulation (3.18) on the joint velocities and accelerations are introduced. Next, criteria based on the movement *along* the path will be discussed, a type of criteria available for both models.

Joint criteria

In [45], a number of criteria for energy minimization in robots are evaluated. The focus is on criteria where a dynamic model for the robot is not necessary. The two criteria which performed best from an energy perspective are the squared weighted acceleration and the squared weighted pseudo power. The former, when used in the kinetic energy formulation (3.18) becomes

$$\frac{c_i(s_i)}{\sqrt{e_i(s_i)}} = \frac{\left(\mathbf{w}_i \circ \ddot{\mathbf{q}}_i(s_i)\right)^T \ddot{\mathbf{q}}_i(s_i)}{\sqrt{e_i(s_i)}}, \tag{3.31}$$

where $\circ$ is the Hadamard product (element wise multiplication) operator and $\mathbf{w}_i$ is a vector of joint weighting factors specific to system i. The choice of weights is discussed later. We note that this expression is convex.

The second criterion, which in [45] resulted in the greatest energy reduction, is the weighted squared pseudo power

$$\begin{aligned}\frac{c_i(s_i)}{\sqrt{e_i(s_i)}} &= \frac{\left(\mathbf{w}_i \circ \dot{\mathbf{q}}_i(s_i) \circ \ddot{\mathbf{q}}_i(s_i)\right)^T \left(\dot{\mathbf{q}}_i(s_i) \circ \ddot{\mathbf{q}}_i(s_i)\right)}{\sqrt{e_i(s_i)}} \\ &= \sqrt{e_i(s_i)}\left(\mathbf{w}_i \circ \mathbf{f}_i'(s_i) \circ \ddot{\mathbf{q}}_i(s_i)\right)^T \left(\mathbf{f}_i'(s_i) \circ \ddot{\mathbf{q}}_i(s_i)\right).\end{aligned} \tag{3.32}$$

The reason behind the nomenclature pseudo power is that the real power is $P = \tau_i \dot{q}_i$, where τ_i is the torque, but in the absence of a detailed dynamic model, acceleration is used as a substitute for the torque. We will denote these two criteria $(\ddot{q})^2$ and $(\dot{q}\ddot{q})^2$ for short.

Because this criterion is non-convex, we choose to apply a zero order approximation for the outer term at a reference velocity resulting in the quadratic criterion

$$\frac{c_i(s_i)}{\sqrt{e_i(s_i)}} = \sqrt{\bar{e}_i(s_i)}\left(\mathbf{w}_i \circ \mathbf{f}_i'(s_i) \circ \ddot{\mathbf{q}}_i(s_i)\right)^T \left(\mathbf{f}_i'(s_i) \circ \ddot{\mathbf{q}}_i(s_i)\right), \tag{3.33}$$

where $\bar{e}_i(s_i)$ is a reference squared path velocity.

Path criteria

In early work on velocity planning, it was observed that dynamically scaling an existing trajectory "allows for modification of movement speed without complete dynamics recalculation" [47]. The dynamic scaling of a trajectory can be regarded as moving along a scaled reference trajectory, and dynamically changing scaling of the reference velocity. We propose penalizing changes to the scaling.

For the kinetic energy model this could be simply the squared path acceleration

$$\frac{c_i(s_i)}{\sqrt{e_i(s_i)}} = e'(s_i)^2. \tag{3.34}$$

and for the inverse velocity formulation (3.29), the squared positional variation of the inverse velocity

$$z_i(s_i)c_i(s_i) = z'(s_i)^2, \tag{3.35}$$

is penalized. We will denote these two criteria path acceleration and pseudo path acceleration respectively, or $(\ddot{s})^2$ and $(z')^2$ for short. While these criteria certainly are simple, it is not quite obvious how they will influence the resulting solution.

Consider the single robot case with bounds on neither velocity nor acceleration. If there are no constraints on initial or final path velocity and the final time is T, then the trivial optimal solution to the two criteria above is to use a constant path velocity $\dot{s}(t) = 1/T$. Constant path velocity incurs zero cost for (3.34) and (3.35), since $\ddot{s}(t) = z'(t) = 0$, and is always feasible due to the lack of bounds. The resulting optimal trajectory is $\mathbf{f}(t/T)$, i.e. the path function, time scaled such that the final time holds. We conclude that if $\mathbf{f}(s)$ originates from a sampled robot motion, then the solution for this unbounded single robot problem is to run a constant time scaled version of that motion.

One could regard the minimization of (3.34) and (3.35) as following a reference trajectory scaled by $\dot{s}(t)$. The cost incurs a penalty on changes in $\dot{s}(t)$, i.e. a cost on changing the scaling of the reference trajectory. When bounds on acceleration and velocity are introduced, $\dot{s}(t)$ may have to vary for the solution to remain feasible. Staying on the velocity and acceleration boundaries is thus penalized and the solution will tend to avoid high acceleration and velocity, i.e. energy efficient rather than aggressive solutions are promoted.

3.1.4 Benchmark

This section will present a case study which evaluates the computational efficiency of the proposed formulations. We have also evaluated the resulting trajectories in an industrial robot This experimental evaluation is presented in Chapter 5. The case consists of two KUKA KR30 6-DoF industrial robots each running the same path, $\mathbf{f}_1(s) = \mathbf{f}_2(s)$ with a single shared zone.

The path is a so called 'pick and place' movement consisting of 458 samples sampled at an interval of 0.012 s. We denote this path the nominal path $\mathbf{f}^n$. A complementary path $\mathbf{f}^c$ was also created using the optimal solution from the weighted squared acceleration minimal solution of one robot performing $\mathbf{f}^n(s)$ with a final time of 8.25 s.

In an industrial setting, it is common for the robot with lower priority to move up to the zone boundary and simply wait before it can gain access. Thus, for comparison, we created an additional trajectory following the same path but with a stop and wait command at the zone boundary, allowing the other prioritized robot performing the original trajectory to clear the intersection zone. The new trajectory resulted in a nominal final time of 7.3 s for the two robots. This is referred to as the nominal two robot solution.

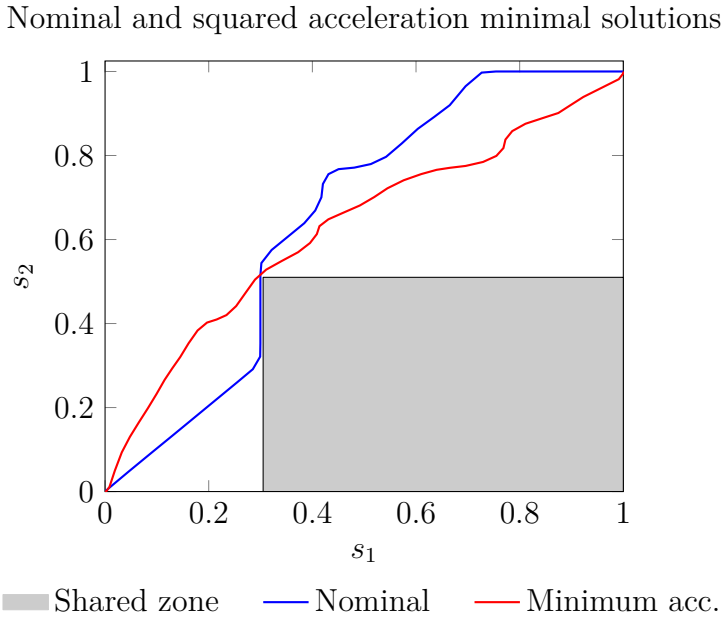

Figure 3.1: The collision set for the two robots (gray), nominal trajectory (blue) and optimal trajectory for the minimum squared acceleration case (red) for the benchmark example with final time 7.3 s.

Figure Figure 3.1 shows the difference between the nominal two robot solution and the minimum squared acceleration solution. Notice how the first robot in the nominal path moves up to the common zone as quickly as possible to the edge of the common zone where it waits for access. The optimized solution shows how the first robot moves through the zone boundary just as the second robot leaves at $t = 3$ s.

The problem was modeled using the AMPL [48] modeling interface, using a zero order hold discretization for the differential equations. For the quadratic program resulting from the inverse velocity formulation (3.29), CPLEX 12.6.3 was used as a solver. For the nonlinear programs, the open source solver IPOPT 3.8.1 [49] was used. The algorithms were run on a Windows 10 64-bit operating system with a 3.29 [GHz] Intel i5 2500K CPU (Q9400). The explicit model is included for comparison, and as it is already very non-convex, the squared joint power criterion is used without zero order approximation.

Table 3.1 contains the recorded computation times. There is not much difference in computation time for the kinetic energy formulation (3.18) when the criteria is varied. The average time was 0.85 s and it performed roughly 8 times faster than the explicit approach used in [45] that averaged 7 s. The inverse velocity formulation (3.29) with the $(z')^2$ criteria is clearly the fastest with an average time of 0.135 s. The inverse velocity formulation was approximately 6 times faster than the kinetic energy formulation and 50 times faster than the explicit formulation. The computation times differ significantly because the inverse velocity problem is a quadratic program rather than a non-convex nonlinear one. Also, not only is the problem class simpler but the software used to solve it is also more mature.

Table 3.1: Computational times summary

Model	Criteria			Runtime [s] with final time: 7.3 s	7.5 s	8.0 s
Kinetic energy (3.18)	Squared joint acc.	$(\ddot{\mathbf{q}})^2$	(3.31)	0.82	0.82	0.73
	Squared joint power	$(\dot{\mathbf{q}}\ddot{\mathbf{q}})^2$	(3.33)	0.79	0.77	0.85
	Path acc. $\mathbf{f}^n$	$(\ddot{s})^2$	(3.34)	0.77	0.85	0.77
	Path acc. $\mathbf{f}^c$	$(\ddot{s})^2$	(3.34)	1.04	1.11	0.87
Inverse velocity (3.29)	Pseudo path acc. $\mathbf{f}^n$	$(z')^2$	(3.35)	0.12	0.11	0.11
	Pseudo path acc. $\mathbf{f}^c$	$(z')^2$	(3.35)	0.16	0.15	0.16
Explicit (3.9)	Squared joint acc.	$(\ddot{\mathbf{q}})^2$	(3.31)	5.07	10.9	6.63
	Squared joint power	$(\dot{\mathbf{q}}\ddot{\mathbf{q}})^2$	(3.32)	6.45	6.52	6.28

3.2 Decomposition approach

The previous section presented a monolithic approach to the acceleration bounded problem with fixed sequences. But as we have seen in Section 2.2 and the benchmark example of the previous section, it is not always necessary to impose timing conditions at every single position along the path. Here, we will decompose the problem by use of parameterization. The following can be regarded as a generalization of the result in Paper 2. Section 3.2.1 will discuss the assumptions made in the paper.

Suppose the evolution of each system i can be divided into ℓ_i stages, defined by the stage transition conditions $s_i(t) = \Delta_i^k$, $k \in \{1, \ldots, \ell_i + 1\}$. A system is in stage k when $\Delta_i^k \leq s_i(t) \leq \Delta_i^{k+1}$. Furthermore, let ν_i^k denote the path velocity at a transition point k, i.e. where $s(t) = \Delta_i^k$. Also let τ_i^k denote the time of transition. Note that $\Delta_i^1 = 0$ and $\Delta_i^{\ell_i+1} = 1$. Figure 3.2 illustrates the use of this notation for a three stage system with transition conditions $\Delta_i^2 = 1/3$ and $\Delta_i^3 = 2/3$.

$$
\begin{aligned}
& e_i^k(\nu_i^k, \nu_i^{k+1}, \tau_i^k, \tau_i^{k+1}) = \min \int_{\tau_i^k}^{\tau_i^{k+1}} c_i(s_i(t))dt \\
& \text{subject to} \\
& \left.\begin{array}{ll} s_i(\tau_i^k) = \Delta_i^k & s_i(\tau_i^{k+1}) = \Delta_i^{k+1} \\ \dot{s}_i(\tau_i^k) = \nu_i^k & \dot{s}_i(\tau_i^{k+1}) = \nu_i^{k+1} \end{array}\right\}, \\
& \left.\begin{array}{l} |\mathbf{f}_i'(s_i(t))\dot{s}_i(t)| \leq \mathbf{v}_i^{\lim}(s_i(t)) \\ |\mathbf{f}_i'(s_i(t))\ddot{s}_i(t) + \mathbf{f}_i''(s_i(t))\dot{s}_i(t)^2| \leq \mathbf{a}_i^{\lim}(s_i(t)) \end{array}\right\} \\
& \qquad\qquad \forall t \in [\tau_i^k, \tau_i^{k+1}],
\end{aligned}
\tag{3.36}
$$

where $e_i^k(\nu_i^k, \nu_i^{k+1}, \tau_i^k, \tau_i^{k+1})$, not to be confused with the $e_i(s_i)$ from the previous section, is the minimum cost as a function of transition velocity and time. Also, let $b_i^k(\nu_i^k, \nu_i^{k+1}, \tau_i^k, \tau_i^{k+1}) \leq 0$ define the feasible set for the above problem. We reiterate that due to time-invariance, only the stage duration is important and as such, $b_i^k(\cdot)$ and $e_i^k(\cdot)$ are in fact only three dimensional.

Given the two functions $e_i^k(\cdot)$ and $b_i^k(\cdot)$, the master problem problem is

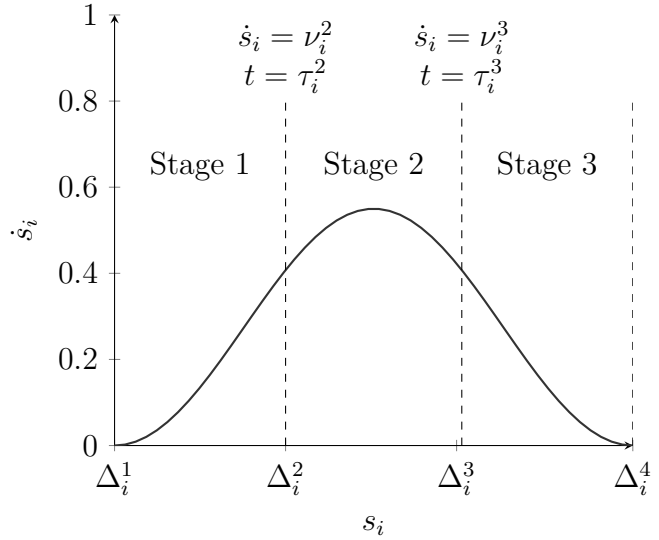

Figure 3.2: The decomposition of a path into a three stage system.

$$
\begin{aligned}
&\min \left[w_1 \max_{i \in \mathcal{N}} (T_i) + w_2 \sum_{i \in \mathcal{N}} \sum_{k=1}^{\ell_i} e_i^k(\nu_i^k, \nu_i^{k+1}, \tau_i^k, \tau_i^{k+1}) \right] \\
&\text{subject to} \\
&\left.\begin{array}{ll} \tau_i^0 = 0 & \tau_i^{\ell_i+1} = T_i \\ \nu_i^0 = v_i^0 & \nu_i^{\ell+1} = v_i^f \end{array}\right\} \qquad \forall i \in \mathcal{N}, \\
&b_i^k(\nu_i^k, \nu_i^{k+1}, \tau_i^k, \tau_i^{k+1}) \leq 0 \quad \forall k \in \{1, ..., \ell_i\} \;\; i \in \mathcal{N}, \\
&\tau_i^k \leq \tau_j^r \qquad \forall \{i, j, k, r\} \in \bar{\mathcal{G}},
\end{aligned}
\tag{3.37}
$$

where $\bar{\mathcal{G}}$ is a set of discretized collision avoidance constraints described by tuples $\{i, j, k, r\}$, i.e. segment k of system i should begin before segment r of system j. The master problem can be summarized as scheduling the transition times and computing the optimal transition velocities.

In summary, the stages within each system primarily interact by continuity conditions on velocity at the switching points, as well as some constraints on the transition times. But at a local level inside each stage, only the duration spent is of importance due to time invariance. The cost incurred during each stage is a function of three external variables, the two boundary velocities and the duration. We express the minimum cost of a stage k of system i as a function $e_i^k(\cdot)$, and the feasible space the external variables as $b_i^k(\cdot)$.

If the functions $e_i^k(\cdot)$ and $b_i^k(\cdot)$ are somewhat well behaved, solving this problem could be simpler than the direct approach. However, one obvious problem is that the functions are unknown to us. We note that our decomposed problem very much resembles that of a nonlinear programming primal decomposition, where the switching velocity and stage durations are global variables, and the dynamics are local. Such an approach is used for a vehicle intersection problem in [50], where the

global variables are the stage switch times, rather than the boundary velocity and stage duration as in our case.

Another approach, which we opted for in Paper 2, is to exhaustively evaluate the functions $e_i^k(\cdot)$ and $b_i^k(\cdot)$ on a grid. This could very well be performed using the mathematical programming approach in the previous section. Unfortunately, when we began working on the decomposition approach, the convexity results of [43], used in the kinetic model of the previous section, had only been known for less than a year and were overseen. Instead we settled for a dynamic programming approach. Since a three dimensional degrees is quite demanding, we opted for a simplified approach, where both the initial and final velocity of each segment were set to zero.

3.2.1 Zero velocity transition condition

At first glance, the requirement of zero transition velocity may seem like an unreasonably restrictive limitation, especially for collision constraints such as the vehicle following example in Section 2.1. But in the case of industrial robots, this is not necessarily the case. When specifying collision zones for industrial robots, one often uses a lot of margin for safety concerns, and the robots may perform large portions of their programs entirely within shared zones. Also, it is not uncommon for the robots to stop, or almost stop, after performing a movement instruction.

With the velocities at the transition points restricted to zero, the minimum cost of (3.36) is now a function of the initial and final time $\bar{e}_i^k(\tau_i^k, \tau_i^{k+1})$. Note that due to time invariance, this is really only a one-dimensional function. Also, let the minimum time solution be denoted δ_i^k. We will not consider any maximum time as there should not be a problem to stop entirely on the time scales considered.

The master problem now becomes simply

$$\begin{aligned}
&\min \left[w_1 \max_{i \in \mathcal{N}} (T_i) + w_2 \sum_{i \in \mathcal{N}} \sum_{k=[1,\ldots,\ell_i]} \bar{e}_i^k(\tau_i^k, \tau_i^{k+1}) \right] \\
&\text{subject to} \\
&\quad \tau_i^0 = 0 \quad \tau_i^{\ell_i+1} = T_i \qquad \forall i \in \mathcal{N}, \\
&\quad \tau_i^k + \delta_i^k \leq \tau_i^{k+1} \qquad \forall k \in \{1, \ldots, \ell_i\} \quad \forall i \in \mathcal{N}, \\
&\quad \tau_i^k \leq \tau_j^r \qquad \forall \{i, j, k, r\} \in \bar{\mathcal{G}},
\end{aligned} \tag{3.38}$$

which is nonlinear in the criterion and linear in all constraints. In Section 4.3, we will present some benchmarks for this problem class. If we were to substitute $\bar{e}_i^k$ for a function based on the linear scaling of an existing velocity profile, we would have the method presented in [12].

3.2.2 Parameterization by dynamic programming

We mentioned previously that dynamic programming can be used to parameterize $\bar{e}_i^k$. There are a large number dynamic programming approaches to the minimum energy problem for single robots with fixed path, beginning with [44] and later iterative

dynamic programming [51]. Most works regarding minimum energy use a weighted energy and time criterion, since excluding an explicit representation of time reduces the state space to a single dimension, velocity. Thus, for the parameterization, a number of problems must be solved with varied weighting between energy and time. In our work, we argued that it is computationally beneficial to explicitly include time, such that a single problem instance is sufficient for parameterization.

In Paper 2, we presented a benchmark using a Matlab implementation run on a Windows 7 64bit system with a 2.66 [GHz] Intel Core2 Quad CPU and 4 [GB] of RAM. Using a resolution of 30 iterations, with a 50×186 size $\dot{s}_i/t$-grid, each stage problem was solved in 40 s. As we three years later compare this to the performance of the monolithic formulation, this clearly is too slow.

Thus, we decided to reimplement the algorithm from Paper 2, this time utilizing an adaptive grid. Furthermore, the boundary of the reachable set was carefully kept track of and was used for grid adaptation, resulting in viable solutions at much lower resolution. Run on the more modern computer used for the monolithic model benchmark, the results indicate a 30 step problem can be solved in less than 4 seconds. This solution time may be even further reducible with an efficient C++ implementation. In retrospect, solving multiple instances of the dynamic programming formulation in [44] may be even faster due to its implementational simplicity. Regardless, this would still most likely prove too slow.

Today, using the convexity results of [43], and repeated solving of a mathematical program or utilizing a multi objective nonlinear programming solver is probably the most efficient way to sample/parameterize the stage costs. In [46], the authors exploit the structure of the resulting single robot problem and report solution times of 10 ms for a 50 step problem. Given such performance, each parameterization problem can be solved in well less than one second.

3.3 Summary

This chapter has discussed mathematical programming models for the continuous trajectory subproblem. That is, the discrete decisions governing the sequence in which robots cross shared zones is known. Two approaches are presented, a monolithic model in Section 3.1 and another decomposition based in Section 3.2.

In the monolithic approach, the continuous problem is encoded into a single NLP using space discretization. Two variable changes found in the literature are utilized, resulting in two models. A variable change based on kinetic energy results in a model where only the collision avoidance timing constraints are nonlinear, and another based on inverse velocity is only nonlinear in the acceleration constraints. The latter is linearized using an inner approximation, resulting in a quadratic program.

For the decomposition approach, the temporal intervals in time instances used for collision avoidance are considered as stages. The inputs/outputs of each stage is the duration and initial/final velocity. The continuous trajectory within each stage is considered as subproblem. The problem is simplified even further by assuming zero initial and final velocity, such that each stage is only a function of its duration. The stage cost is parametrized using dynamic programming.

Chapter 4

Sequencing problem

In the previous chapter, it was assumed that the sequence in which the robots move through the shared zones is known, i.e. which robot is prioritized. Here, the sequence is now once again assumed to be unknown. Thus, instead of linear inequalities, the collision zones are modeled by a disjunctions between linear equalities. In other words, the robot priority is now a discrete decisions which must be made.

Whereas the previous chapter was focused on modeling improvements, this chapter is concerned with algorithmic improvements. When encoded into a mathematical program, the disjunctive inequalities are converted into mixed integer expressions. The nonlinear problem formulation in the previous chapter is now a mixed integer nonlinear program. Recall that algorithms for MINLPs are discussed in Section 2.3.

Both problem formulations in the previous chapter could be characterized as scheduling problems with additional nonlinear constraints. In this chapter, we will examine the integration of constraint programming with mathematical programming methods for such problems. We will show that a straight forward implementation provides good results, even against algorithms which already include such integration.

4.1 Integrated optimization

We will attempt to improve the performance of the MINLP algorithms discussed in Section 2.3 by including components from constraint programming. Since the early 2000s [52], constraint programming and mixed integer linear programming methods have been combined to solve scheduling problems more efficiently. A survey on the use of integrated methods in scheduling is presented in [53].

A popular approach is to decompose the problem using logic-based Benders Decomposition [40]. For example, in minimum time resource assignment problem without precedence constraints, a master integer programming problem will specify the assignment problem and constraint programming is used to solve the resulting allocation problem. The result from the constraint program will then be entered into the master integer program as a linear cut. In this particular case that approach works very well, as the lack of precedence constraints allows each resource to be treated as a separate problem, and as such the cuts generated by the constraint programming become much stronger. This works very well in the absence of precedence constraints

as the scheduling of operations on each resource is independent of one another. In the more general setting, including precedence constraints, more complex performance criteria, etc, such a decomposition is not as efficient.

Another approach which incorporates linear programming into a constraint programming solver can be found in [54]. In contrast to Logic based Benders decomposition, which uses a top layer of integer programming and constraint programming for subproblems, the approach in [54] includes both constraint and linear programming models of when entire problem is created. At each node in the constraint programming branch and bound tree, a corresponding linear programming relaxation of the current node is created. The solution of the relaxation provides a bound on the objective function.

An even tighter integration of constraint and mathematical programming is SIMPL [55]. It uses its own modeling language to specify the optimization problem, after which the solver decides itself which methods should be invoked to solve the problem. Integrated constraint and mixed integer linear programming can also be found in for example IBM ILOG CP Optimizer, which uses both in its large neighborhood search [56].

Finally, as previously mentioned, both Baron [57] and SCIP (Solving Constraint Integer Programs) [58] utilize constraint propagation and can solve non-convex MINLPs to global optimality. It is not clear how extensive the use of constraint propagation in Baron is, but in SCIP it is an integral part of the solver. Our work differs in that SCIP and Baron are both used for the global solution of non-convex MINLPs, while ours is streamlined for convex formulations.

4.1.1 Proposed implementation

In short, our proposed algorithm uses branch and bound, and at each node, a portfolio of algorithms is available. The portfolio considered in this paper consists of constraint, linear and nonlinear programming. Each algorithm in the portfolio uses its own model and requires an interface to communicate any branching decisions or variable bound reductions to the algorithm.

Any constraints which are not available/suitable for an algorithm are ignored, e.g. nonlinear constraints in constraint programming or global constraints (e.g. disjunctive) in mathematical programing. Thus, since the cost in our problems is based on nonlinear functions, the constraint programming part cannot tell us anything about the cost, only feasibility with regards to the other constraints. That is, the constraint programming model includes only a subset of the original problem, i.e. the traditional scheduling part. The real valued variables must also be discretized to some appropriate resolution.

The algorithm computationally cheapest to execute, is constraint programing, which may be used to determine whether a node is feasible or not. Next is the linear program, which provides a lower bound to the solution in the current branch. Finally, the nonlinear program not only provides lower bounds, but if we are at a leaf (terminal) node, it also yields an upper bound to the global solution. In addition to

solving subproblems, we have the additional choice of adding linearizations to the linear program, to improve its lower bound.

The overall execution of the integrated algorithm is described in Algorithm 2. In essence, the algorithm can be regarded as Algorithm 1 with the addition of constraint propagation at the point before linear programing is performed, i.e. at the stage where cutting planes are applied. For the implementation in this thesis, Gecode 4.4.0 [41] is used for constraint propagation, Clp [59] for the linear programs and Ipopt [49] for the nonlinear programs. These algorithms are used in a modular fashion, where the input is the vector of binary variables and the output is the result of the algorithm. Tightened variable bounds may also be communicated from the constraint propagation phase. For the linear program, there is also the choice of adding linearizations.

Since constraint propagation is computationally inexpensive to perform, it should in general be performed in every node. Also, since constraint propagation is also likely to detect scheduling infeasibilities before linear or nonlinear programing, it should also be performed first, in the hope that we need not run the other algorithms. In such a setting, it is possible to realize the integrated algorithm design by embedding the mathematical programming part into a constraint propagator.

Recall the execution of a constraint programming algorithm described in Section 2.3.2. First, all propagators are scheduled for execution. After execution, unless the propagator deems the current node infeasible, a propagator is either taken of the list of scheduled propagators, or it is rescheduled. The propagators are typically scheduled in order of computational requirements, beginning with the lowest complexity propagators.

Suppose we design a new constraint for which we of course need to implement a propagator. The constraint is applied to the binary variables as well as our cost variable. The propagator of this new constraint is scheduled for execution whenever there is a change to the binary variables of the problem. Furthermore, the cost of execution for the propagator is set to the maximum, effectively scheduling it last of all propagators. In such a setting, at each node, the constraint programming algorithm will perform propagation as usual, but before completing, it will also execute the code inside our own propagator. This opens up for a possibility to run mathematical programming, and modify the domain of the cost variable accordingly.

If steps 7-10 of the Algorithm 2 are embedded inside the custom propagator discussed in the previous paragraph, we can see that the overall behaviour of a constraint program with the added custom constraint will be that of Algorithm 2. This approach removes the need to write any branch and bound code since one already exists in the constraint programming algorithm. We have successfully applied this embedding approach using both the open source constraint programming solver Gecode in [60] and the commercial solver Ilog CP Optimizer in [20].

4.2 Benchmark problems

To make a thorough benchmark, we will generate a large number of problem instances. Here, we will define two generic problems based on the formulations in Section 3.1

Algorithm 2 Integrated Algorithm

1: **Initialize.**
Populate the search tree Γ with the root node, initialize the linear programming MP, initialize the upper bound $c_{ub} = c_{max}$.
2: **Terminate?**
If the search tree is empty, $\Gamma = \emptyset$, terminate.
3: **Select.**
Select the next node γ from Γ.
4: **Schedule propagators.**
Any propagators subscribing to the variable domain subject to the branching is added to the list of scheduled propagators.
5: **Select propagator.**
Select the next propagator from the list, if list is empty goto 7.
6: **Propagate**
Execute the propagator, if the constraint is infeasible, goto 2. If any variable bounds were decreased, schedule any available propagators subscribing to the concerned variables. The current propagator is either flagged as available or unavailable for rescheduling. Goto 5.
7: **Evaluate LP.**
Solve linear MP based on the full/partial integer assignment γ, if infeasible goto 2.
8: **Prune.**
If the current solution $c_{MP}(\gamma) \geq c_{ub}$, goto 2.
9: **Solve NLP?**
If we are at a leaf node (full integer assignment), solve the NLP corresponding to T, else goto 12.
10: **Upper bound?**
If the NLP solution is feasible and cost $c_{NLP}(\gamma) \leq c_{ub}$, update the upper bound.
11: **Refine.**
Add linear approximations to the MP based on the NLP solution, goto 2.
12: **Divide.**
Branch on current node, add the new nodes to the search tree Γ, goto 2.

and Section 3.2. For both models, we will also include a mixed integer quadratically constrained quadratic program (MIQCQP) version, such that we can include an additional algorithm to our comparison.

4.2.1 Multiple robot path coordination

The first problem is based on the monolithic space formulation (3.8) in Section 3.1. We will use a simplified version where the robots are single degree of freedom rotating arms and rotate from one point to another without stopping. This makes the velocity and acceleration constraints linear in (3.8). The minimization criteria is the squared acceleration.

We note that in contrast to (2.12), this particular benchmark examples has used

$$(t_j(g_{ij}(s_i)) \leq t_i(s_i) \vee t_i(g_{ji}(s_j)) \leq t_j(s_j)) \quad \forall s_j \in [0,1]. \tag{4.1}$$

This constraint differs from (2.12) in that $s_j \in [0,1]$ is placed outside the disjunction. When $s_j \in [0,1]$ is placed inside, there is clearly only one alternative. However, when it is placed outside, each discretization point of s_j inside the collision zone constitutes an alternative, although implicitly, all of these apparent decision are one and the same.

Since the sequence is now unknown, the collision avoidance constraint is now posed including the choice of priority as in (2.12). The collision avoidance zones for rotating single degree of freedom robot arms is defined by (2.17).

Recall that the problem is non-convex due to the time constraint $t_i'(s_i) = 1/\dot{s}_i(s_i)$. In this benchmark we would like to work with convex problems, and also pose a MIQCQP for comparison. Thus, we will relax this constraint such that

$$t_i'(\theta_i) \geq \frac{1}{\omega_i(\theta_i)}, \tag{4.2}$$

which is convex, since the velocity is non-negative. From our experience, at least for the special case of unbounded acceleration, this lower bound is tight. Regardless of tightness, the relaxation allows us to apply convex methods and include state of the art MIQCQP methods in our benchmark.

4.2.2 Nonlinear job shop scheduling

The second problem is a mix of the decomposition approach (3.38) in Section 3.2 and a traditional job shop scheduling problem. In a job shop scheduling problem, there are n different jobs that are to be scheduled on m different machines, indexed by $\mathcal{M} = [1, .., m]$. Each jobs consists of a set of operations which are to be processed in a given order, each on a predefined machine. Each operation also has a fixed processing time. Furthermore, a job does not use the same machine twice, operations may not be preempted, and each machine can only process one job at a time.

The constraints describing the system consist of

$$t_{ij} + d_{ij} \leq t_{i+1,j} \qquad \forall i \in \mathcal{N} \;\; j \in \mathcal{M} : \;\; i \neq n, \tag{4.3}$$

$$t_{nj} + d_{nj} \leq T \qquad \forall j \in \mathcal{M}, \tag{4.4}$$

$$t_{ij} + d_{ij} \leq t_{kl} \vee t_{kl} + d_{kl} \leq t_{ij} \qquad \forall \langle i, j, k, l \rangle \in \mathcal{D}, \tag{4.5}$$

where t_{ij} and d_{ij} are the starting time and duration of operation j in job i, $j \in \mathcal{M}$ and $i \in \mathcal{N}$, T is the final time (makespan) of the system and $\mathcal{D}$ is a set of tuples $\langle i, j, k, l \rangle$ defining each pairs of operations ij and kl processed on the same machine. As for the constraints: (4.3) describes the given order of operations within a job; (4.4) ensures the final time is larger than each job completion time; (4.5) finally ensures

that each machine only processes one job at a time. If the final time is minimized, the decision variables of the problem becomes the final time T and the starting times t_{ij}, while the operation durations d_{ij} each have a minimum duration δ_{ij}.

In a mixed integer program, the logical constraint (4.5) will have to be converted to mixed integer form using the big-M technique (not to be confused with initial point generation in linear programs). That is, (4.5) becomes

$$t_{ij} + d_{ij} \leq t_{kl} + M(1 - b_{ijkl}) \qquad \forall \langle i, j, k, l \rangle \in \mathcal{D}, \tag{4.6}$$
$$t_{kl} + d_{kl} \leq t_{ij} + M b_{ijkl} \qquad \forall \langle i, j, k, l \rangle \in \mathcal{D}, \tag{4.7}$$

where $b_{ijkl} \in \{0, 1\}$ is a boolean variable which is 1 when operation ij precedes kl, and 0 if kl precedes ij, and M is a constant large enough to relax the inactive constraint. If the final time is upper bounded by T_{ub}, then $M = T_{ub} + d_{ij}$ in (4.6), and $M = T_{ub} + d_{kl}$ in (4.7). Note that in the constraint programming, the mutual exclusion constraints (4.5) can be replaced, or supplemented for additional tightness, with the global constraint (2.21).

To mimic the type of problem resulting from the decompositional approach, we add an energy minimization term. Each term is a cubic expression on the form

$$f_{ij}(d_{ij}) = 4(d_{ij} - \delta_{ij})^2/\delta_{ij} + (d_{ij}/\delta_{ij})^3, \tag{4.8}$$

where f_{ij} is the cost term, δ_{ij} the constant minimum duration and d_{ij} is the variable duration of operation ij. Note that the expression is convex for $d_{ij} \geq 0$. The cost term represents the energy consumption of an operation. The constraints of the nonlinear job shop problem are defined by (4.3)-(4.8) and the criterion is the sum over all terms in (4.8).

Even though the problem belongs to the MINLP class, it is in fact possible to reformulate this model as a mixed integer quadratically constrained program, and solve it using for example Cplex, Gurobi and Express. We will later include Gurobi [61] in our benchmark, and thus present the following quadratically constrained version.

The cubic term d_{ij}^3 can be modeled by first introducing an over approximator of the squared duration

$$d_{ij}^{sq} \geq (d_{ij})^2, \tag{4.9}$$

and consequently an over approximator for the cubic term

$$d_{ij}^{cube} d_{ij} \geq (d_{ij}^{sq})^2. \tag{4.10}$$

which can be modeled using modern MIQCQP solvers, as these typically accept constraints on the form $x^2 \leq yz$, where $y \geq 0$ and $z \geq 0$.

4.3 Benchmark

Using the two problem classes introduced in Sections 4.2.2 and 4.2.1, we benchmark our proposed implementation as well as a number of existing MINLP implementations.

Two convex MINLP solvers are included. The commercial solver Knitro includes three algorithms for MINLP, one of which is a variation of LP/NLP-BB [62]. The open source solver Bonmin, which features a number of algorithms, one of which is a variation of LP/NLP-BB. Out of the box, Bonmin uses the open source software Cbc [63] for mixed integer linear programming and Ipopt [49] for its NLP subproblems.

Two global non-convex MINLP solvers are also included. Baron is a commercial solver where constraint propagation is utilized, although to what extent is unclear [57]. For non-convex problems it uses a spatial branch and bound algorithm based on linear programs for bounding. The open source solver SCIP [58] is also evaluated. Similarly to Baron, it uses LP based bounding via spatial branch and bound. SCIP integrates constraint and mathematical programming and has support for global constraints.

Finally, the commercial MIQCQP solver Gurobi is also included in our benchmark [61]. A thorough review of MINLP software is available in [64].

4.3.1 Multi robot path coordination

Three problem sizes were considered for the multi robot path coordination problem, the number of robots $n \in [2, 3, 4]$. For each problem size, 100 instances were generated with a randomized initial position and rotation direction. The total distance rotated was set to one revolution and the robots were positioned in two rows, close enough for collisions to occur, the total time was set to $T = 5$ s. A discretization grid of 40 points was used and a forward Euler approximation was used for the continuous dynamics. Also, recall that the constraint propagation methods work with integer variables. Thus the variable domains are upsampled, we chose a factor 20, making the smallest time unit 0.05 s. A lower or higher upsampling factor would result in a faster or slower results, respectively.

At *each* discretization point, a disjunctive timing constraint (4.1) would be posed, each with a unique boolean variable. Recall the discussion from Subsection 4.2.1, that this does introduce some redundant boolean variables, even though there are only two alternatives for each collision.

In addition to the standard variation of our algorithm, denoted *Integrated*, a variation dubbed *Integrated w/o LP* was also used in this example. In this addition, the linear approximation part has been disabled such that the algorithm exclusively uses constraint propagation in the tree and NLPs at the nodes. The reason to exclude linear approximations from the search is that these simply do not result in any bounded nodes due to the weak relaxations. In practice, this implies that constraint propagation is used to find scheduling feasible solutions, and NLP in turn is used to compute the optimal cost.

The right hand side of Figure 4.1 shows an overview of the result. The solver Bonmin would abort solving for many instances. These have been marked as unsolved.

Integrated w/o LP clearly outperforms all other algorithms, even Gurobi. *Integrated* algorithm also performs very well. Only three algorithms solves all problems within the time limit: *Integrated*, *Integrated w/o LP* and SCIP, all of which use constraint propagation technology. Baron performs the worst, only solving problem instances which have no feasible solution. However, we suspect Baron erroneously treats the problem as non-convex, which could make the search much more difficult. The commercial MINLP solver Knitro does rather well, but not nearly as good as *Integrated*, *Integrated w/o LP*, SCIP and Gurobi.

Note that using *Integrated w/o LP* in favour of *Integrated* sped up the problem simply by reducing the number of unnecessary LP solved. A similar phenomenon was observed in Paper 5, where two instances of a similar, but non-convex problem was studied. Due to the non-convexity, iterative linearization could not be used and standard nonlinear branch and bound was employed. Thus, each additional node explored was very expensive, since an NLP had to be solved in each node. To overcome this, the following solution was devised. In addition to the method suggested here, after step 6 of Algorithm 2, before any NLP was solved, two child nodes would be created and constraint programming would be used to search for feasible solutions. An NLP would only be solved if both child nodes contained feasible solutions, this would greatly reduce the number of NLPs due to redundant boolean variables.

The results of Paper 5 are in line with what we have observed in this benchmark. When the subproblems are large, there is more time to gain from, and more time to actually run, constraint propagation methods. The constraint propagation approach even outperform Gurobi, the quadratic programming solver. For very difficult subproblems, explicitly finding all the integer feasible solutions and solving these individually, without hope of bounding by cost, may be a good strategy.

4.3.2 Nonlinear job shop scheduling problem

For the NJSSP, we considered four sizes, $n \in \{4, 5, 6, 7\}$, with $m = n$. For each problem size, three different final times were considered, $T \in \{T_{min}, 1.1 \cdot T_{min}, 1.2 \cdot T_{min}\}$, where T_{min} is the minimum final time. For each of these $4 \times 3 = 12$ configurations, 100 instances with different operation times were randomly generated from a uniform distribution. The solver Baron crashed for problems of size 6 and 7 with $T = 1.2 \cdot T_{min}$. Also SCIP would crash, although for problems of size 7 with $T = 1.2T_{min}$. These instances were marked as unsolved.

An overview of the results can be found in the left hand side of Figure 4.1. Our proposed algorithm is denoted *Integrated*. It uses the disjunctive constraint (2.21) for additional tightness. We have also included results for our algorithm without this additional constraint, where the result is denoted *Integrated w/o disj.*. Clearly, Gurobi is the fastest across almost all instances. Recall that Gurobi is a MIQCQP solver and its performance should be regarded as an estimator of achievable performance by MINLP algorithms.

For the instances with $T = T_{min}$, our implementation is clearly faster than the other MINLP algorithms. The other MINLP solvers including constraint propagation

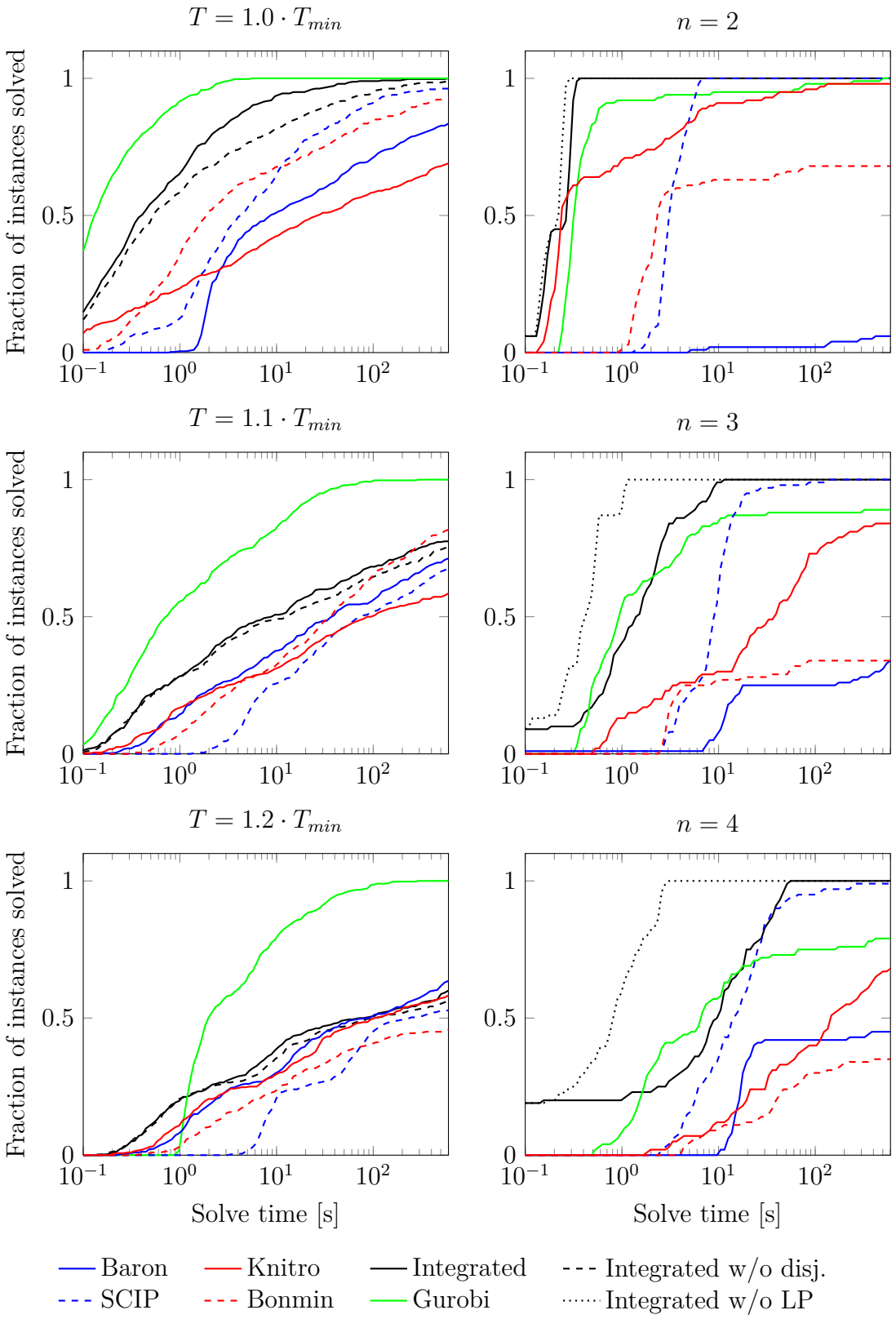

Figure 4.1: The left most figures show the fraction of nonlinear job shop instances solved vs solution time ($T \in \{T_{min}, 1.1 \cdot T_{min}, 1.2 \cdot T_{min}\}$, where T_{min}), while the right most figures show fraction of multi robot coordination instances solved vs solution time ($n \in \{2, 3, 4\}$)

technology (Baron and SCIP) follow. The traditional MINLP solvers are the slowest. These instances typically have a low number of feasible solutions, making the constraint propagation methods likely to reduce the solution time.

For the medium cycle time instances $T = 1.1 \cdot T_{min}$, our implementation is fastest for almost all instances but the most difficult, for which Baron has a slightly higher success rate. The rest of the algorithms achieve very similar benchmarks.

Finally, for the high cycle time instances $T = 1.2 \cdot T_{min}$, the traditional MINLP algorithms perform very similar to our implementation. These instances are characterized by a very large number of feasible solutions to search. This is rather naturally explained by the fact that the constraint propagation methods find less infeasibilities due to cycle time constraints, and thus create an overhead for the algorithm rather than a speedup.

The performance of our proposed method seems to degrade for the larger instances. It seems that when there are a large number of feasible solutions, the constraint propagation methods struggle to improve the search. At the same time, our algorithm lack many of the features included in the comparison methods, such as strong branching. This may begin to play a more important role for larger instances.

4.4 Summary

In this chapter, we have looked closer at the discrete sequencing problem. And specifically, how the problem may be solved more efficiently by used of constraint propagation techniques. Existing methods and our proposed approach were discussed in Section 4.1. We also define two benchmark problems: the multi robot path coordination problem in Section 4.2.1 based on the monolithic approach in Section 3.1; and nonlinear job shop scheduling in Section 4.2.2 inspired by the decomposition approach in Section 3.2.

In a thorough benchmark, we compared our proposed approach to existing MINLP methods. When a problem instance is characterized by difficult subproblems, or when the number of feasible solutions is restricted by a near minimal final time, our proposed approach outperforms existing software.

Chapter 5

Experimental results

As we have previously mentioned, the primary application for our methods is that of industrial robots. This chapter will present experimental results which will indicate the potential of energy savings. We will begin by describing the experimental setup, whereafter the experimental results are presented, followed by a brief analysis.

5.1 Experimental setup

We have previously stated that detailed system parameters are unknown and thus, torque constrains cannot be included. In the absence of torque, it becomes crucial to directly constrain angular velocity, acceleration, and jerk by suitable values so that the robot's envelope of operation is not violated. The values depend on the positional configuration of the joints at each time instance, which is referred to as pose or sample of the path. In practice, we infer the bounds in all poses from the velocity, acceleration, and jerk values calculated along the original trajectories.

Furthermore, up till now, jerk constraints have been ignored. We offer two options to overcome this. The first is to extend the explicit formulation (3.9) with jerk. Although it is already non-convex and will be even more so with jerk constraint, the problem can still be solved since in practice, the original trajectory is available as a feasible initial solution. The second option is to post process the result. If we allow the fixed path condition to be relaxed, a jerk bounded quadratic reference following problem can be posed. We have used both these methods with success, the latter resulted in a maximum path deviation in the range of $1 - 2$ degrees.

During operations, the robots warm up and the resistors inside the servo motors' circuits exhibit more resistance. At the same time the lubrication system in the joints becomes more efficient, resulting in lower friction in mechanical components. The net effect is a significantly decrease in power consumption as the robot warms up. Therefore, it is very important to conduct all measurements at the same realistically high temperature to obtain reliable and replicable results.

A program was written to warm up the robot to the desired set-point, run the experiments sequentially, and warm up/cool down according to the measured set-point deviation in between experiments. Half an hour of warm up can reduce

the energy consumption more than 20%, and steady state temperature may not be reached for hours.

The energy has been measured either at the power cords, leading to the robot's servo motors, or at the power source. In the latter case, the cabinet power was estimated and subtracted from the measurements. Energy was measured directly using Chauvin Arnoux PAC22 current clamps and Testec SI 9002 differential voltage probe. The data was sampled using Data Translation DT9826 acquisition box operating at 10 kHz. Active powers at each sample were calculated by direct multiplication of voltage and current samples, and energy was calculated by integration of active power over time.

5.2 Results

This section will review the most important energy reduction results.

Two robot minimum energy

Let us begin with the energy measurements from the case study described in Subsection 3.1.4. For this experiment we use the KUKA KR 30-3, a medium payload robot with a maximum capacity of 30 kg.

The results are summarized in Table 5.1. Both joint based criterion, available to the kinetic energy based model (3.18) performed well, i.e. squared acceleration (3.31) and pseudo-power (3.33). These used roughly 22% less than the nominal case (using the same final time). The pseudo power criteria performed slightly better than the acceleration criterion.

The two path based criteria did not perform as well as the squared acceleration and pseudo power. The path criterion for the kinetic energy model, $(\ddot{s})^2$, reduced energy by 17.3% and 17.6% depending on the path function ($\mathbf{f}^n$, $\mathbf{f}^c$). And the path criterion $(z')^2$, for the inverse velocity model (3.29), reduced energy by 15.9% and 18.6%. Overall, $(\ddot{s})^2$ performed marginally better than $(z')^2$. This could either be due to the lack of available acceleration from the inner approximation of acceleration bounds in the inverse velocity model, or simply due to some property inherent to z'. The choice of path function for the two path criteria $(\ddot{s})^2$ (kinetic model) and $(z')^2$ (inverse velocity model) did not have much affect on the energy consumption. The complementary path $\mathbf{f}^c$ based on the minimum acceleration trajectory used slightly less energy.

Note that Table 5.1 does not present any measurements for the explicit formulation (3.9). This is because the criteria used in the explicit formulation are nearly identical to those of the kinetic energy formulation. The only difference is the zero order approximation of the velocity term in (3.32), which did not impact the energy consumption noticeably. Thus, the measured energy is expected to be very much the same.

To summarize, minimizing squared acceleration or pseudo power, available to the kinetic energy formulation, provides the most energy efficient results. The second model which uses an inverse velocity variable change is not as energy efficient,

Table 5.1: Criteria/time: two robot example

Model	Criteria			Energy reduction [%] with final time: 7.3 s	7.5 s	8.0 s
	Nominal $\mathbf{f}^n$			0	2.8	7.9
Kinetic energy (3.18)	Squared joint acc.	$(\ddot{\mathbf{q}})^2$	(3.31)	21.3	24.2	28.9
	Squared joint power	$(\dot{\mathbf{q}}\ddot{\mathbf{q}})^2$	(3.33)	21.5	25.3	29.4
	Path acc. $\mathbf{f}^n$	$(\ddot{s})^2$	(3.34)	17.3	19.7	23.7
	Path acc. $\mathbf{f}^c$	$(\ddot{s})^2$	(3.34)	17.6	20.8	26.4
Inverse velocity (3.29)	Pseudo path acc. $\mathbf{f}^n$	$(z')^2$	(3.35)	15.9	18.5	24.2
	Pseudo path acc. $\mathbf{f}^c$	$(z')^2$	(3.35)	18.6	20.9	26.2

with reductions of 15.9%/18.6% compared to the 22.5% achieved by minimizing the acceleration or pseudo power. The inverse velocity formulation is however computationally much faster to solve.

Single robot peak power

Using the same medium size robot as in the previous example, we have studied the effect of constraining the peak pseudo-power of the entire trajectory. At first, the trajectory was optimized without any regard to the power. Next, the peak pseudo-power was calculated for the optimized trajectory, and then a constraint was added to the model bounding the peak pseudo-power using the calculated value. Next, in steps of 5% length, the bound was tightened. At 50% of the original peak pseudo-power limit, the problem became infeasible.

The results are depicted in Figure 5.1. As can be seen, tightening the constraint on mechanical pseudo-power directly reduces the measured electric active power. At its extreme, almost 25% peak power reduction is produced at the cost of only 3% energy consumption. We conclude that by adding a peak power penalty to the problem, one may significantly reduce peak power without losing too much energy efficiency.

Multiple robots minimum energy

To test the optimization method on a realistic production cell with complex trajectories, we used a robot cell located at Daimler AG. It comprised of four KUKA KR 210 QUANTEC Prime robots, each capable of handling 210 kg of payload. The cell was powered by dc grid, and all the equipment, including robots and welding guns were dc operated.

The robots would assemble a sheet metal part of a car body and the scenario is as follows: (i) the first robot, designated by R10, picks up three parts and applies glue to them; (ii) the parts are placed in a fixture, where the second robot, R20, rivets them together; (iii) the third robot, R30, picks up the assembled part from the fixture, marks it and places it in another fixture for welding; (iv) the fourth

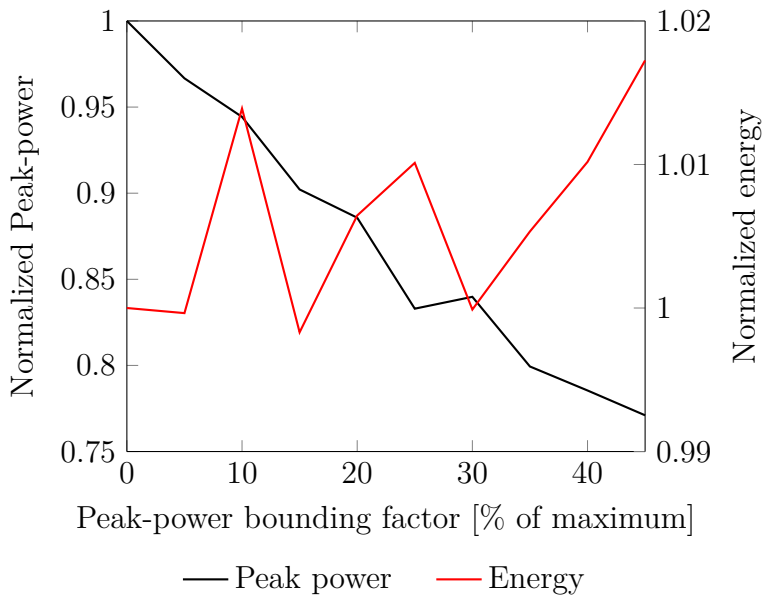

Figure 5.1: Trade-off between peak power (black) and energy (red).

robot, R40 performs the welding, and then the parts are placed by R30 in a bin. The robot programs of this scenario were written by technicians at Daimler AG, and included an array of motion types such as linear, point-to-point, and circular. Moreover, the points were programmed with a variety of accelerations, velocities, and approximation factors.

The production engineers at the cell had imposed the requirement that the cell had to produce a part every $T = 68$ s. In practice, the cycle time of the cell, including the overhead time of communications with the programmable logic controller and resetting the robots, surpassed the requirement, at about 64 s. Furthermore, three to four seconds was overhead time, therefore, the actual motions took about 60 s. Robot R10 was the bottle-neck and governed this cycle time.

We observed that it was possible to relax the cycle time constraints on robots R10 and R30 without exceeding the imposed cycle time of 68 s. Hence the two robots received 4.5 and 10 seconds of extra time in the optimization. Note that large portions of the velocity profiles were fixed, such as where welding was performed. Only 32% of R10's motion and 52% of R30's motion were allowed to be changed. The results are summarized in Table 5.2, which also contains measurements from the KUKA robots' internal energy saving feature that operates based on down-scaling velocity and acceleration. These are labeled Eco Low, Eco Middle and Eco High

Typically, the motors in an individual robot are all connected to local robot dc bus. If one motor breaks, the negative breaking power may be used to power another motor, with some losses of course. Any surplus energy, which is not used by another motor, is burnt as heat through a resistor. In this particular experiment, since all robots were connected to a larger dc grid, surplus energy could be fed back into the grid for use by other robots.

The results are summarized in Table 5.2, where E_{gross}, the amount of energy that a robot consumed from the grid, E_{regen} is the amount of surplus energy fed back into

Table 5.2: Results of the multi-robot case.

Robot	Scenario	T_i	E_{gross}	E_{regen}	E_{net}	P_{max}[kW]	$E^{\downarrow}_{\text{gross}}$[%]	$E^{\downarrow}_{\text{net}}$[%]	$P^{\downarrow}_{\text{max}}$[%]
R10	Original	60.0	91.7	12.4	79.3	19.8	0.0	0.0	0.0
	Optimized	64.5	69.8	5.3	64.5	8.0	**23.9**	**18.7**	**59.4**
	Eco Low	61.9	83.7	9.7	74.0	18.6	8.7	6.7	6.2
	Eco Middle	63.7	79.3	7.7	71.6	15.7	13.5	9.7	20.7
	Eco High	66.1	75.5	6.8	68.7	13.9	17.6	13.3	29.9
R30	Original	40.5	88.0	17.4	70.6	22.9	0.0	0.0	0.0
	Optimized	50.8	62.8	4.6	58.2	7.7	**28.6**	**17.6**	**66.5**
	Eco Low	42.4	85.8	15.8	70.0	18.0	2.5	0.8	21.5
	Eco Middle	44.1	80.4	13.0	67.4	17.1	8.6	4.5	25.2
	Eco High	45.4	77.1	10.5	66.6	16.1	12.4	5.6	-29.8

the grid. That is, in a traditional setup, the robot would consume E_{gross}, but this particular setting, the total energy consumed is $E_{\text{net}} = E_{\text{gross}} - E_{\text{regen}}$. Moreover, P_{max} is the peak active power. Finally, the percentage of energy savings are given based on E_{gross} and E_{net}.

As seen in the table, the optimization resulted in significant reduction of energy consumption and peak-power. This was achieved while total cycle time of the cell was still below the required 68 s. The energy reduction for R30 was slightly better than that of R10, possibly since more of its motion was subject to optimization. In comparison, the results obtained from robots' internal energy saving function, namely Eco Middle, gave similar cycle time, but yielded significantly less savings, especially in terms of peak power. Note that this feature works on the principal of reducing speed and acceleration based on some heuristics. Better results were obtained from Eco High mode, but the total cycle time of the cell then violated the mentioned requirement.

5.3 Analysis

Here we will take a closer look at the measurements by model fitting. Using the result, we will attempt to offer a hypothesis for why squared acceleration and squared pseudo power work well for energy reduction. For notational simplicity, the robot index i will be omitted. The torque vector for a robot can be expressed by a Lagrange formulation [65]

$$\boldsymbol{\tau} = J(\mathbf{q})\ddot{\mathbf{q}} + C(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + F_s(\mathbf{q})\dot{\mathbf{q}} + F_v(\mathbf{q})\text{sign}\,(\dot{\mathbf{q}}) + G_i(\mathbf{q}), \tag{5.1}$$

where $J \in \mathbb{R}^{m\times m}$ is the inertia matrix, where $m \in \mathbb{Z}^+$ is the number of joints, $C \in \mathbb{R}^{m\times m}$ the matrix of centrifugal and Coriolis coefficients which is linear in the joint velocities, $F_s \in \mathbb{R}^{m\times m}$ the coulomb friction matrix and $F_v \in \mathbb{R}^{m\times m}$ the viscous friction matrix, $G \in \mathbb{R}^m$ the gravitational vector.

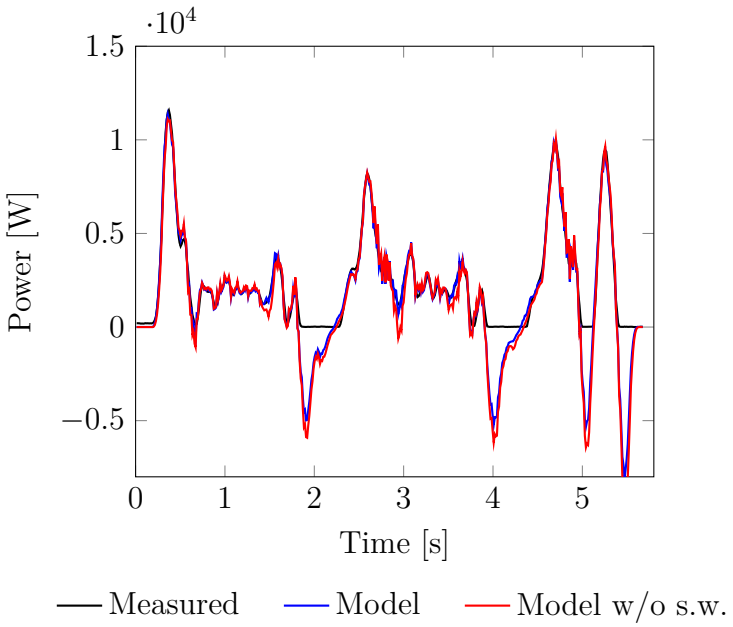

Figure 5.2: Measured power (black) and model fit with (blue) and without stator winding (red).

Considering the relevant time scales, as in [66] and [67] , the energy consumption of an ac permanently excited synchronous motor can be expressed by the following simplified voltage and dc models

$$\boldsymbol{\nu}(t) = R \circ \boldsymbol{\iota}(t) + K_V \circ K_R \circ \dot{\mathbf{q}}(t), \tag{5.2}$$

$$\boldsymbol{\tau}(t) = K_T \circ K_R \circ \boldsymbol{\iota}(t), \tag{5.3}$$

where $\circ$ once again is the Hadamard (element wise) product, $\boldsymbol{\iota}(t) \in \mathbb{R}^m$ and $\boldsymbol{\nu}(t) \in \mathbb{R}^m$ are the equivalent direct current and voltage vectors, $R \in \mathbb{R}^m$ the stator resistance, $K_V \in \mathbb{R}^m$ the electrical (back emf) constants, $K_R \in \mathbb{R}^m$ the transmission gear ratio and $K_T \in \mathbb{R}^m$ the equivalent torque constant. With (5.2)-(5.3) defined, we can express the power of each motor as

$$\mathbf{p}(t) = \boldsymbol{\nu}(t) \circ \boldsymbol{\iota}(t). \tag{5.4}$$

The KUKA robots provide estimates for the individual motor currents, and since the joint velocities are known, we can combine the voltage equation (5.2) with the power equation (5.4) such that

$$\mathbf{p}(t) = R \circ \boldsymbol{\iota}(t) \circ \boldsymbol{\iota}(t) + K_V \circ K_R \circ \dot{\mathbf{q}}(t) \circ \boldsymbol{\iota}(t), \tag{5.5}$$

for which we can make a least squares estimator. However, the measured power only includes the positive side of the summed powers over all motors, i.e.

$$\max\left(0, |\mathbf{p}(t)|_1\right). \tag{5.6}$$

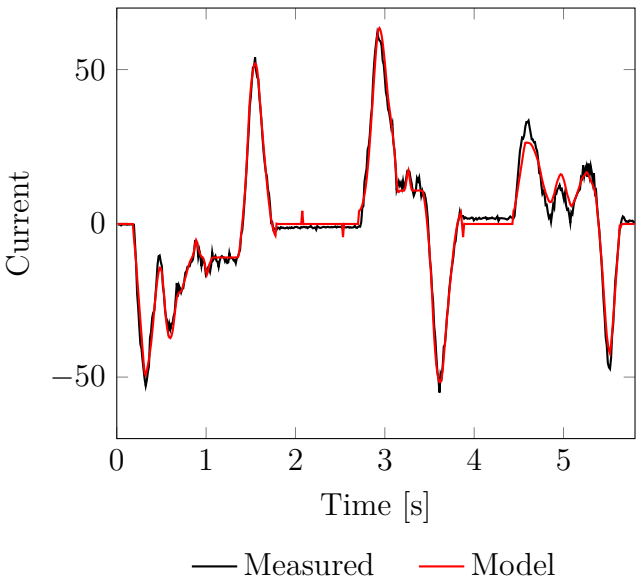

Figure 5.3: Measured (black) and estimated (red) current in join one.

Thus any samples with lower measured power than some threshold are discarded. Also, the data is synchronized to ensure the best fit. The resulting least squares estimate for pick and place motion used in the two robot example is illustrated in Figure 5.2.

In the two robot experiment, the motors on a robot all share a local dc bus, where any negative power is converted to heat using a resistor. Note that before converted to heat, a small portion may be stored in a capacitor for future use, this is not included in our model. Using our least squares estimator we can now visualize this negative power. Furthermore, it turns out that the most important term in (5.5) is $\dot{\mathbf{q}}(t) \circ \boldsymbol{\iota}(t)$, i.e. the stator heat loss R is not very important.

We can also create a least squares estimator for the current by substituting the torque (5.1) into the current (5.3), while assuming constant inertia, friction, gravity, etc. To reduce the degrees of freedom, we will also assume the joint torques are decoupled. If we disregard the velocity component of the matrix of centrifugal and Coriolis coefficients we get an expression on the form

$$\iota_j = \alpha_1 \ddot{q}_j + \alpha_2 \dot{q}_j + \alpha_3 \text{sign}\left(\dot{q}_j\right) + \alpha_4, \tag{5.7}$$

where ι_j is the current of motor $j \in \{1, .., m\}$, q_j is the motor position and a_i are constant coefficients to be estimated. Even this very simplified model fits the data surprisingly well, see Figure 5.3 for an illustration of the first join current and model fit. Note that we do not suggest this approach for system identification purposes but rather for an indication of what occurs inside the system. We find that the acceleration coefficient α_1 dominates the other terms, even after normalization.

To summarize, the (5.5) is dominated by $\dot{\mathbf{q}}(t) \circ \boldsymbol{\iota}(t)$, and (5.7) by $\ddot{q}_j$. If we combine this information it is clear why minimizing the pseudo-power $\dot{\mathbf{q}}(t) \circ \ddot{\mathbf{q}}(t)$, yields the best results in our experiments.

5.4 Summary

The focus of this chapter has been the experimental evaluation of the proposed models in the previous chapters. We have described the experimental setup, presented results and provided a short analysis of measured data. We can conclude that in the presence of slack time, i.e. when the robots are not fully utilized, there most definitely is potential for energy saving. As for quantification, the amount saved clearly depends on how the slack time can be spread around the different operations.

In the two robot example, we demonstrated energy reduction of $20 - 30\%$, depending on the amount of slack time. In the multi robot case, we saw that one of the robots would not utilize all of its slack. That is, there are diminishing returns from increased slack. For this particular example, the energy savings are also in the range of $20 - 30\%$.

Chapter 6

Hybrid systems

Recall from Section 2.2 how the structure of our problem resembles very much that of synchronized multi stage systems. In this chapter, we will regard our problem from an even broader perspective, and consider the case of hybrid systems with shared variables. We will discuss what sort of limitations our problem class imposes and how our methods would work for the hybrid systems case. Specifically, we focus on the multiple hybrid systems case and the modeling of communication between systems. The scope of this chapter is quite limited, and we will make use of simple examples to illustrate how multiple systems may be modeled by mathematical programming, and the subsequent problems that ensue.

6.1 Hybrid optimal control

Optimization methods for hybrid systems aim to compute optimal or sub-optimal trajectories for the systems. A hybrid trajectory can be regarded as a mode (discrete state) sequence and a set of continuous state trajectories, one for each mode in the sequence. In the general case, the transition timing and sequence cannot be determined a priori. The combinatorial explosion in possible mode sequences combined with the infinite-dimensional trajectories in each mode makes the problem very difficult so solve [68]. Unless some simplification or abstraction is made, a general hybrid optimal control problem usually result in a non-convex optimization problem.

There are several approaches based on variations of dynamic programming, as in [69]–[71]. Common in these approaches is that while dynamic programming guarantees global optimality, it unfortunately suffers from the 'curse of dimensionality', because the method involves computing a large number of trajectories, and the problem grows exponentially in the number of states.

Another method, involving the extension of classical optimal control, is the hybrid maximum principle, used in for example [72]–[74]. The maximum principle can be considered as a set of necessary conditions for optimality, given a hybrid optimal control problem. For some simple cases, it may be used to carry out analytical calculations of the optimal solution, but in general, numerical methods must be

employed. The resulting formulation is solved using a multiple-phase multiple-shooting method [75]. That is, combinatorial algorithms are used to enumerate a neighbourhood of mode trajectories, for each mode trajectory, the optimal continuous-time state trajectories are determined using numerical methods. The numerical methods generally do not guarantee global optimality.

For the solution of the maximum principle, necessary conditions can be summarized as a two stage approach. A neighborhood of mode sequences are enumerated and for each sequence, the continuous dynamics are considered. In a similar fashion, hybrid optimal control problems may be monolithically expressed as a non-convex MINLP [68], [76] or Generalized Disjunctive Program [77]. A nonlinear program is used to approximate/collocate the continuous time dynamics [78], and the mixed integer part is used to model the mode switching behaviour. A search algorithm then iterates over the neighbourhood of hybrid mode trajectories while considering the continuous-time dynamics in each iteration. In both the maximum principle and MINLP approaches, the neighbourhood of mode trajectories must be defined a priori.

If the system is affine or well approximated as piecewise affine, the whole hybrid optimal control problem could be modeled in time with a fixed sample length as a mixed integer quadratic program [79]. At each time instance, boolean variables specify the active mode of the current sample. This enables both proof of global optimality and modeling of systems with a high number of switches. Other advantages of the piecewise affine approach include the maturity of mixed integer quadratic solvers, results for the system closed loop behaviour [80], [81], and furthermore multi-parametric mixed integer linear programming solvers may be used to generate control laws for closed loop control [82]. In the case of multiple systems, expressing the global mode by binary variables at each time instance may be intractable.

In summary, numerical approaches based on dynamic programming guarantee global optimality, while increasing very quickly in complexity as the problem size grows. Other numerical methods either sample uniformly in time, or utilize the fact that the problem may be hierarchically decomposed into a discrete mode sequencing problem and a continuous multi stage optimal control problem (MSOCP). The decomposition approach, to which our methods lie closest, lack proof of global optimality and may have difficulties in treating systems with a high number of mode switches. Amongst the decompositional approaches, the main difference lies in the type of model and methods used for the sequencing and optimal control problems.

6.2 Hybrid systems with shared variables

The study of hybrid systems has resulted in a large number of modeling formalisms. Here, we will use the hybrid automaton [83] [84] to describe the individual hybrid systems. Hybrid automata are generalized finite-state machines with the usual discrete transitions as well as continuous state dynamics which may vary with each mode (discrete state). The continuous states may also influence the discrete transitions.

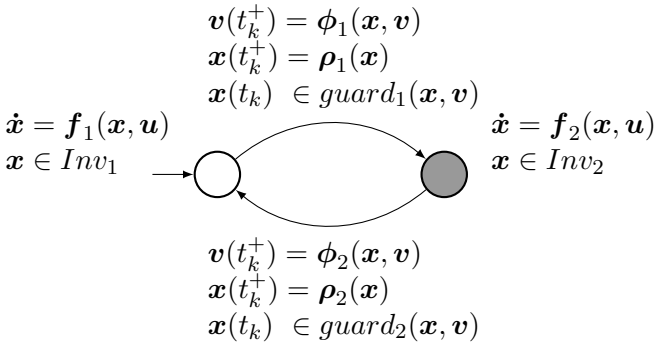

Figure 6.1: Two mode hybrid system. The initial mode is marked by an incoming arrow, and although we have not explicitly included any final state in the model, the final mode is indicated by gray color.

We will now consider shared variables as a means of communication between the individual systems [85], and as such extend the hybrid automaton with a set of shared integer variables, separate from the normal state variables. Note that the shared variables are only updated during mode transitions.

Based on the notation in [84], we define an individual hybrid automaton with shared integer variables as

$$G = \langle \mathcal{Z}, \mathcal{V}, \mathcal{X}, \mathcal{U}, \boldsymbol{f}, \boldsymbol{\phi}, Inv, guard, \boldsymbol{\rho}, z_0, \boldsymbol{x_0} \rangle, \tag{6.1}$$

where $\mathcal{Z} = \{z_1, ..., z_m\}$ is a set of discrete states or modes, $\mathcal{V}$ is the domain of the shared integer variables, $\mathcal{V} \subseteq \mathbb{Z}^{n_v}$, $\mathcal{X}$ is a continuous state space, $\mathcal{X} \subseteq \mathbb{R}^{n_x}$, $\mathcal{U}$ is a set of admissible input signals, $\mathcal{U} \subseteq \mathbb{R}^{n_u}$, f is a vector field, $\boldsymbol{f} : \mathcal{Z} \times \mathcal{X} \times \mathcal{U} \to \mathcal{X}$, $\boldsymbol{\phi}$ is a discrete state/variable transition function $\boldsymbol{\phi} : \mathcal{Z} \times \mathcal{X} \times \mathcal{V} \to \mathcal{Z} \times \mathcal{V}$, Inv is a set defining an invariant condition, $Inv \subseteq \mathcal{Z} \times \mathcal{X}$, $guard$ is a set defining a guard condition, $guard \subseteq \mathcal{Z} \times \mathcal{Z} \times \mathcal{X} \times \mathcal{V}$, $\boldsymbol{\rho}$ is a reset function, $\boldsymbol{\rho} : \mathcal{Z} \times \mathcal{Z} \times \mathcal{X} \to \mathcal{X}$, z_0 is the initial discrete state, v_0 is the initial value of the shared variables, and $\boldsymbol{x_0}$ is the initial continuous state. Figure 6.1 provides a graphical representation of the model. Note that we have excluded events from the formulation as we consider shared variables a sufficient means of communication.

The vector field $\boldsymbol{f}$ describing the dynamics of $\mathcal{X}$ takes the form

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(z(t), \boldsymbol{x}(t), \boldsymbol{u}(t)), \tag{6.2}$$

for all times $t \neq t_k$, where $t_k : \ k = \{1, 2..\}$ are the time instances when transitions occur, and $z(t)$ is the mode active during t. At each discrete transition, the reset condition $\boldsymbol{\rho}$ updates the continuous state vector as

$$\boldsymbol{x}(t_k^+) = \boldsymbol{\rho}(z(t_k^+), z(t_k), \boldsymbol{x}(t_k)), \tag{6.3}$$

where t_k is an arbitrary time when a discrete transition occurs. The discrete mode and shared integer variables are updated as

$$\begin{bmatrix} z(t_k^+) \\ \boldsymbol{v}(t_k^+) \end{bmatrix} = \boldsymbol{\phi}(z(t_k), \boldsymbol{x}(t_k), \boldsymbol{v}(t_k)). \tag{6.4}$$

A transition at time t_k is possible when the $guard(z(t_k^+), z(t_k), \boldsymbol{x}(t_k), \boldsymbol{v}(t_k))$ is satisfied.

Let us discuss these variable updates in more detail. For our purposes, the shared variables act much like resources which are incremented at allocation and decremented when deallocated. For example, an allocation of a shared variable v_ℓ is stated simply $v_\ell(t_k^+) = v_\ell(t_k) + 1$. The update conditions on a shared variable will disable a transition if its new value, $v_\ell(t_k^+)$, should lie outside its domain. This definition of synchronization is somewhat informal, but it will suffice for the purposes of this thesis. For notational simplicity we introduce the short hand notations v_ℓ^+ and v_ℓ^-, equivalent to $v_\ell(t_k^+) = v_\ell(t_k) + 1$ and $v_\ell(t_k^+) = v_\ell(t_k) - 1$ respectively

6.2.1 Multi robot coordination case

Let us shortly discuss where the problem formulation in Chapter 2 fits into this. Recall the discretization of collision zones as in Figure 2.5 in the same chapter. Here, we will consider the collision zones to be discretized in such a way. Also, for notational simplicity, we will not use the system index i.

Each phase, i.e. the phase in-between synchronization points, is represented by a discrete mode, and each system visits its mode in sequence. The mode q is always incremented since the modes are visited in sequence. For each collision zone a shared discrete variable v_ℓ is introduced. These variables are binary since a collision zone may only be occupied by one robot at a time. Thus, the discrete variables in (6.4) are updated at transition time t_k as

$$\begin{bmatrix} z(t_k^+) \\ v_\ell(t_k^+) \end{bmatrix} = \begin{bmatrix} z(t_k) + 1 \\ v_\ell(t_k) \pm 1 \end{bmatrix}, \tag{6.5}$$

where $\pm$ indicates that either an increment or a decrement is applied to the ℓ:th shared variable v_ℓ, $\ell \in \{1, \ldots, n_v\}$. Note that the mode update is independent of the continuous state. The discrete mode update is illustrated in Figure 6.2, where p_j, $j \in \{1, 2, 3, 4\}$, are the zone boundaries. Note that p_j is scalar since only the scalar path position states is relevant to the transition into and out of shared zones, the velocity state is simply subject to continuity conditions.

As for the continuous states, the vector field (6.2) describing the continuous state dynamics is a simple double integrator

$$\dot{x}_1(t) = x_2(t) \qquad \dot{x}_2(t) = u(t), \tag{6.6}$$

while the more complex dynamics are placed inside the invariant Inv and take the form

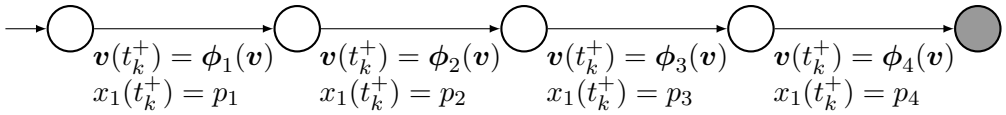

Figure 6.2: The mode sequence, variable update and guard conditions in the multi robot case using a discretized collision set. The differential equation, path constraints and reset condition remain constant. Note that only the scalar path position state is subject to guard conditions.

$$|\mathbf{f}'(x_1(t))x_2(t)| \leq \mathbf{v}^{\lim}(x_1(t)), \tag{6.7}$$

$$\left|\mathbf{f}'(x_1(t))u(t) + \mathbf{f}''(x_1(t))x_2(t)^2\right| \leq \mathbf{a}^{\lim}(x_1(t)), \tag{6.8}$$

where $\mathbf{v}$, $\mathbf{a}$ and $\mathbf{f}$ are the velocity and acceleration constraints, and path function, just as before. The reset conditions (6.3) are simple continuity conditions

$$\boldsymbol{x}(t_k^+) = \boldsymbol{x}(t_k). \tag{6.9}$$

Note that the state equation, invariant and reset conditions are all independent of the active mode.

The mechanism which couples the continuous and discrete dynamics is placed in the guard conditions. The guard conditions generally specify subsets of the continuous state space $\mathcal{X}$ which, when entered while the system mode is $z(t_k)$, enables a transition from $z(t_k)$ to some new mode $z(t_k^+)$ through $\boldsymbol{\phi}$. Since the modes are visited in sequence, the guard conditions take the form

$$(z(t_k) = z_j) \wedge \left(z(t_k^+) = z_{j+1}\right) \wedge (x_1(t_k) = p_j)\,, \tag{6.10}$$

which models that a transition from mode j to $j+1$ is allowed when the position state is p_j, i.e. p_j defines the boundary of the collision zone.

6.3 Mathematical modelling

In this section, we will attempt to fit our ideas for coordination of robots to the coordination of general hybrid systems. In particular, we focus on numerical issues in direct methods in presence of synchronization. In the study of hybrid systems, most often a monolithic model structure is considered. Many of the well known modeling formats restrict the domain of the hybrid mode to only one dimension. For example, the unified hybrid modeling framework in [69] considers the mode domain as the integer numbers, and the mode in hybrid automata is defined as to take values in a set [83]. Other closely related modeling frameworks such as switched and piecewise affine systems are also monolithic in their modes.

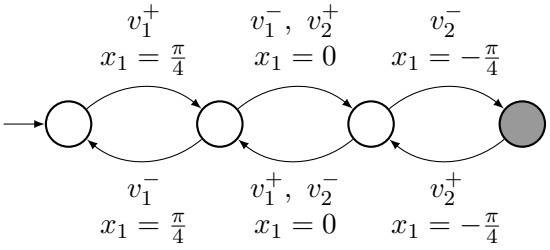

Figure 6.3: A 1-DOF robot arm with two shared zones represented by the variables v_1 and v_2, and a booking procedure ensuring a collision free system.

There are of course a number of extensions which include synchronization, as well as practical implementations. In [86], [87], hybrid dynamical systems are modeled by Petri nets, the continuous dynamics are restricted to single-integrators. Hybrid input/output automata use shared actions for discrete communication and shared variables for continuous communication [88]. Another example is CHARON [89], a language for hierarchical specification of interacting hybrid systems. There are several other examples, most of which are related to analysis.

Within the scope of this chapter, it will suffice to regard synchronization at an informal level. For two systems, the composition of the two systems is a single monolithic system which describes the discrete and continuous dynamics of both. Let us illustrate composition using the robot arm example in Figure 2.5 of Chapter 2, where two robots are moving from a top to a bottom pointing position. The resulting hybrid automaton of the left hand robot is illustrated in Figure 6.3, with its resulting three zone transitions. Note that since the state x now explicitly represents position, we have transformed the boundary from the path domain $[0, 1]$ to angular domain $[\pi/2, -\pi/2]$. The right hand robot would of course have slightly different transition conditions due to its orientation.

We begin with the continuous dynamics. Suppose the system dynamics are identical and described by

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -d\cos(x_1) + u \end{bmatrix}, \tag{6.11}$$

where the constant d is a term describing the mass, gravity and length of the arm. The composed state vector is now of size 4 and is described by the dynamics

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} x_2 \\ -d\cos(x_1) + u_1 \\ x_4 \\ -d\cos(x_3) + u_2 \end{bmatrix}. \tag{6.12}$$

As for the discrete dynamics, the composed mode transition graph of two such robots is illustrated in Figure 6.4. The mode combinations not allowed by the shared variables have been marked with red crosses and transitions removed. The transition labels have been removed for readability.

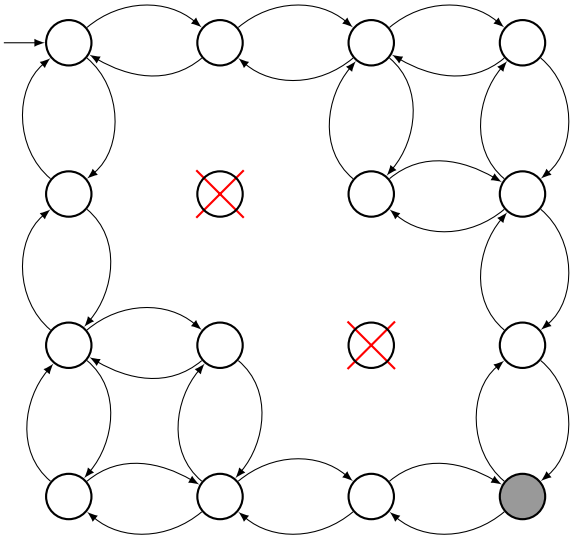

Figure 6.4: Mode transition graph for two composed 1-DOF robot arms.

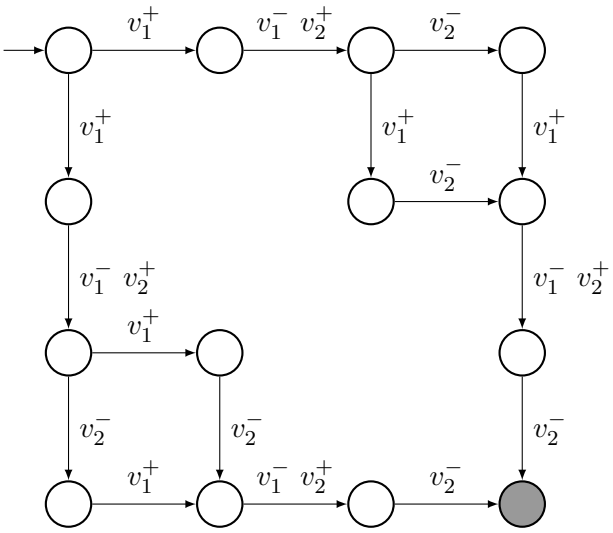

Figure 6.5: Two 1-DOF robot arms with finite mode sequences.

6.3.1 Mixed integer formulation

Search methods such as MINLP require the mode set of possible mode sequences to be finite. And in any case, general direct methods approach the problem by enumeration of possible mode sequences. A general approach could be to define a neighbourhood of discrete mode sequences. For example by defining the maximum times a mode may be visited, and using reachability information to prune unnecessary modes.

In our example, let us simply consider that the robots always move towards their goal state. Although this is a highly restrictive assumption from a modeling sense, we do this because we are interested in illustrating the numerical properties of the resulting composed model. Figure 6.5 shows the resulting composed system, with

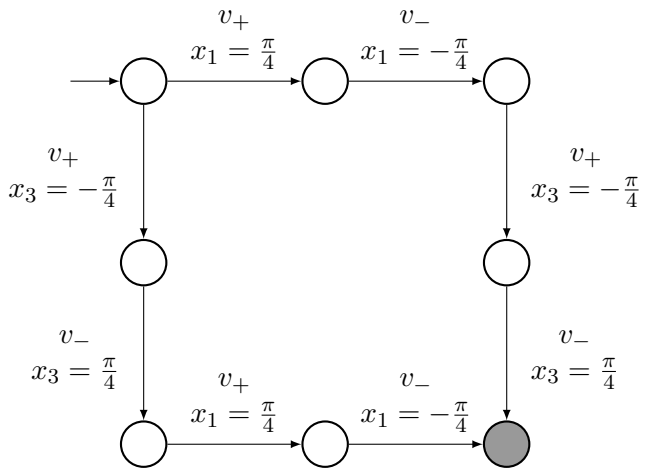

Figure 6.6: Two 1-DOF robot arms with state vectors x and z moving in opposite directions with finite mode sequences and a single shared zone, $x_1, x_3 \in [-\pi/4, \pi/4]$.

the collision states now completely removed.

We can see that even though there is only a single decision to be made, the transition graph of the composed system has introduced additional alternatives into our model. The reason for this, is that the order in which the prioritized robot leaves v_2 and the non-prioritized robot enters v_1 is explicitly kept track of. We conclude that when composing two systems, one should prune these additional alternatives.

Let us consider the even simpler example of where the robots move in opposite directions and must cross each other. The collision zone is as the intersection in Figure 2.3 of Chapter 2, where only one single shared zone is required. The resulting composed transition graph is illustrated in Figure 6.6. There are still redundant transitions, i.e. the initial and final transitions, although there are only two alternatives.

But how should we go about modeling this in a MINLP? Not only should we encode the valid mode sequences, but we must also regard the resulting relaxed problem. We will now review a few different approaches. All models include a boolean variable for the discrete choice, which system is prioritized.

A straight forward approach is to, for each mode in the transition graph, instantiate variables modeling the continuous states. The boolean imposes the boundary constraints and cost contribution of the chosen path, and relaxes the inactive mode sequence. When the boolean is relaxed, all boundary constraints as well as the cost contribution are relaxed. The resulting model consists of 8 modes, 4 of which are redundant, the continuous state vector is of size 4. Denote this the *full model*,

Because the system dynamics are identical and the mode sequence lengths are the same, there is also the option to reuse variables. That is, we create a 5 stage model with a continuous state vector of size 4, where the boolean variable as before switches the correct boundary constraints on and off, i.e. either the top or bottom sequences' boundary constraints hold. Just as before, no boundary constraints hold when the boolean is relaxed, but there is a chain of state continuity constraints linking the initial and final states. Let us denote this approach the *reuse model*.

Another option, is not to compose the systems, but rather impose timing conditions based on the shared variables, similar to the collision zones in the previous chapters, i.e. using a mutual exclusion style constraints. This type of method has previously been used for converting traditional finite automata to mixed integer linear programs [90], [91]. The continuous states are approximated separately for each system, i.e. the model contains two MSOCPs, each with 3 stages and a state vector of size 2. This is denoted the *modular model*.

Finally, if each alternative is examined individually, the monolithic and even the modular models use an unnecessary large number of stages. This is because when a problem is encoded into a MINLP, the size must be fixed. This implies that the total number of stages in each node of the branch and bound algorithm never change, no matter if it is a relaxation or a leaf node. If the size could change size, it would be possible to express these nodes using fewer stages, we denote this the *minimal model*.

Consider the modular model when the sequence is fixed. The two systems actually only interact at one point, the inequality relating the exit from the zone by one robot and the entering of the other. This problem could be modeled using only two stages per system, removing the transition where the robot with priority moves into the zone, and the transition where the last robot moves out of the zone. For the relaxed problem only one stage is required, as the two systems never interact.

6.3.2 Continuous dynamics

As previously mentioned, the evaluation of a mode sequence for one hybrid system corresponds to solving a MSOCP, where each stage corresponds to a mode. The MSOCP, when posed as an NLP, consists of three types of elements: transition timings, boundary conditions and an approximation of the continuous dynamics.

By expressing the time spent (duration) in each mode as variables, we can express the time at any transition as a sum of durations. Thus, we can ensure that the total time spent in modes corresponds to any final time we may have. Boundary constraints ensure either continuity in the continuous states between modes, or that any transition or reset conditions associated with the transitions holds. The system dynamics are then typically modeled using collocation [92] or a shooting method [75]. In Chapter 3 we settled for a simple forward approximation.

In collocation, the continuous states and inputs are approximated as polynomials. These polynomials are differentiated and at selected points, the continuous dynamics are then enforced at these points using nonlinear equalities. Another approach is shooting methods, where the integration of the dynamics is approximated using numerical methods. In summary, the collocation methods result in a large sparse NLP for which the second order information is readily available. The shooting methods are much smaller in size, but it is somewhat harder to compute the second order information.

Let us take a closer look at collocation using the method in [92], and what form the approximation takes for a single mode. Let $L_\ell(t)$, $\ell \in \{0, ..., n_\ell\}$, be a basis of Lagrange polynomials [93], on the interval from $[-1, 1]$ given by

$$L_\ell(t) = \prod_{\substack{i=0 \\ i\neq\ell}}^{n_\ell} \frac{t - t_i}{t_\ell - t_i}, \qquad \ell = [0, ..., n_\ell], \tag{6.13}$$

where we see that

$$L_\ell(t_i) = \begin{cases} 1 & \text{if } \ell = i \\ 0 & \text{if } \ell \neq i \end{cases} \tag{6.14}$$

The state $x(t)$ is approximated by $X(t)$ as

$$x(t) \approx X(t) = \sum_{\ell=0}^{n_\ell} x_i(t_\ell) \cdot L_\ell(t). \tag{6.15}$$

Recall that the polynomial is defined on the interval $[-1, 1]$. The interpolation points used are the boundary point, -1, and the n_ℓ Gaussian quadrature points t_ℓ, $\ell = \{1, \ldots, n_\ell\}$. Thus, by (6.14)

$$X(t_\ell) = x(t_\ell) \quad \ell = \{0, ..., n_\ell\}. \tag{6.16}$$

where $t_0 = -1$. Differentiation of (6.15) yields the following expression for the approximated state derivatives $\dot{X}(t_i)$

$$\dot{x}(t_i) \approx \dot{X}(t_i) = x(-1) \cdot \bar{D}_i + \sum_{\ell=1}^{n_\ell} x(t_\ell) \cdot D_{i\ell}, \tag{6.17}$$

where $D_{i\ell} = \dot{L}_\ell(t_i)$ and $\bar{D}_i = \dot{L}_0(t_i)$.

We can approximate a state equation $\dot{x} = f(x)$ for a duration T using

$$X(-1) \cdot \bar{D}_i + \sum_{\ell=1}^{n_\ell} X(t_\ell) \cdot D_{i\ell} = \frac{T}{2} f(X(t_i)). \tag{6.18}$$

The terminal states are enforced by quadrature approximation of the dynamics as

$$X(1) = X(-1) + \frac{T}{2} \sum_{\ell=1}^{N} w_\ell \cdot f(X(t_\ell)). \tag{6.19}$$

where w_ℓ are gauss weights.

6.3.3 Numerical issues

By varying the grid resolution $n_\ell \in \{5, 7, 10, 15, 25\}$ and the model parameter $d \in \{1, 3\}$, 10 problem instances are generated. The criteria were defined as the sum over integral squared control inputs. The models posed in the previous section are evaluated as NLPs with the binary variables either fixed or relaxed, i.e. what a MINLP algorithm would solve at the leaf and branch nodes respectively. Additionally, a MINLP model is included, using a big-M formulation for the disjunctive constraints.

Using the big-M technique for nonlinear equality constraints may cause numerical issues. One way to overcome this is to pose a generalized disjunctive program (GDP),

which includes the modeling of disjunctive constraints. During branch and bound, only the appropriate constraints are added to the problem. We also pose NLPs where only the relevant constraints are added, we denote this a GDP style model, as opposed to a MINLP style model.

The problem instances were modeled using the AMPL modeling language and solved using, IPOPT [49] as NLP solver and Bonmin [94] as MINLP solver, Table 6.1 contains the NLP results. As the three models differ in size, to make the comparison more fair the analysis will primarily focus on the number of iterations rather than the computational time. In problem instances without numerical issues, changing the grid resolution n_ℓ in the collocation did not have any noticeable impact on the number of iterations.

As expected, the full model proves to have a bad lower bound, while all the other models share the same lower bound. Both the full and reuse model suffer from numerical difficulties when modeled using a MINLP formulation. This is somewhat alleviated when changing to a GDP style formulation. The modular model suffers from small numerical problems in the relaxed setting, and the minimal model none. The latter performed the best on all scales.

When the complete MINLP was solved in Bonmin, the reuse model actually failed to find a solution due to an NLP iteration limit of 3000. If we exclude $n_\ell = 25$, the full model was solved with an average runtime of 4.7 s. The modular model on the other hand solved the instances with an average runtime of 0.2 s.

Even though this is only a small example, it illustrates the importance of considering numerical issues in the relaxations when modeling a MINLP. The results indicate that there is much performance to gain by creating an entirely new model at each node, that is tailored to reduce numerical issues and produce a good lower bound. We conclude that it is not straight forward to encode multiple hybrid systems into a MINLP. Even a single hybrid system which includes alternatives, as opposed to a MSOCP, can easily become problematic.

6.3.4 Constraint propagation

The evaluation of a candidate sequence implies solving an NLP, and can be considered relatively expensive. Clearly it is important to select a search method which evaluates as few solutions as possible. In the case of multiple systems, constraint propagation could be used similarly to Chapter 4, to reduce the number of NLPs solved. Although since the problem is now non-convex, linearization cannot be used to approximate the problem. When the number of feasible integer solutions is low, it may even be beneficial to enumerate all feasible solutions and examine separately.

The approach in Chapter 4 could be modified as follows. At each branch node, a depth search is performed to ensure that a feasible solution exists further down the tree before solving any NLP. The feasibility check was performed using constraint programming, with a simple lower bound on the mode duration, which was independent from the state vector due to lack of bounds on velocity and acceleration. The algorithm would in practice use constraint programming to find feasible solutions, and then apply NLP for that mode sequence. This is explored in Paper 5.

MINLP style model

	Feasible NLP		Relaxed NLP		
	$E[t]$	$E[iter]$	$E[t]$	$E[iter]$	$LB(d=3)$
Full	5.7*(2.3*)	777*(183*)	0.3	13	0
Reuse	5.6*(3.2*)	714*(363*)	32.6*	1563	40.65
Modular	0.11	24	0.16(0.07)	53(21)	40.65

GDP style model

	Feasible NLP		Relaxed NLP		
	$E[t]$	$E[iter]$	$E[t]$	$E[iter]$	$LB(d=3)$
Full	0.30*	41*	0.01	3	0
Reuse	—"—	—"—	12(3.8)	1939(444)	40.65
Modular	0.19	32	0.15 (0.11)	37(19)	40.65
Minimal	0.07	18	0.015	5	40.65

Table 6.1: Average solution time and iteration count for the different models. Parenthesis indicate result after $2-3$ outliers were removed, $*$ indicates that one or two infeasible instance were removed. All models have the same solution for the feasible sequences.

A more elaborate version might use some form of reachability analysis to ensure feasible solutions exist. For example, constraint propagation methods for floating point variables are a maturing technology, which may at some point be used for this purpose. And constraint propagation for quadratic constraints are being used for the global solution of quadratic satisfaction problems [95].

6.4 Summary

In this chapter we have examined our collision avoidance scheme as a synchronization methods for hybrid system. We have discussed what differs our problem from the general case and how the general problem can be encoded into a MINLP. Using a small example, we have illustrated how a bad MINLP encoding can easily cause numerical issues. Preferably, an entirely new NLP, tailored to reduce numerical issues, should be posed at each node in a branch and bound search. For problems with few valid mode sequences, using search methods to explicitly enumerate valid sequences. may prove a tractable approach.

If the problem is serial in nature, as is for example our systems which progress through a series of zones, transforming the system into a multi stage problem could be beneficial. However, the problem of robust solvers for these non-convex problems remain. In other cases, where the discrete mode has a switching nature, a time discretization with binary variables expressing mode occupancy is most likely the best option.

Chapter 7

Stacker crane scheduling

This chapter is concerned with the stacker crane scheduling problem, which is somewhat different from the problem formulation discussed so far. Stacker cranes are moving devices used in material handling systems for the safe storage and retrieval of goods. They are typically used in situations where large volumes of goods must be continuously moved in and out of temporary storage. In short, the stacker crane problem requires us to distribute a set of tasks amongst a set of cranes, decide the order in which the tasks are processed, and finally compute the trajectory of each crane. We would like to find the minimum time solution to the problem.

In [96], a cargo assembly planning problem is solved using constraint programming, and part of the problem consists of pairs of stacker cranes moving along the same rail. Here, we extend the problem formulation to include multiple cranes, performing tasks with both pick up and drop off, as well as bounded acceleration. We also introduce a simplification which reduces the complexity of the problem with very small effect on the solution quality.

7.1 Problem formulation

A problem instance consists of m tasks, each defined by two locations, one for pick up and another for drop off location. Let the constant vector $\mathbf{p} \in \mathbb{R}^{2m}$, indexed by $\mathcal{M}_2 = \{1, \ldots, 2m\}$, describe these $2m$ locations. The first m elements are the pick up and the last m elements are the drop off locations. Also, let the variable vector $\mathbf{a} \in \{1, \ldots, n\}^m$, specify which crane is allocated to handle each of the tasks.

For notational simplicity, we introduce two operators $(\cdot)^+$ and $(\cdot)^-$, such that p_j^+ is the pick up location and p_j^- is the drop of location for each task $j \in \mathcal{M} = \{1, \ldots, m\}$, i.e. $p_j^+ = p_j$ and $p_j^- = p_{j+m}$. For further notational simplicity, let us also introduce a lazy modulo notation where, any index is subject to a modulo operation based on the dimensionality of the vector, such that $a_k = a_{k+m}$.

In this particular problem formulation, movement tasks cannot be preempted. Once a container is picked up by a stacker crane it is required that the container is dropped off in the designated position, and nowhere else. This implies that stacker cranes cannot drop off a container part of the way to its destination and then use a second stacker crane to relay it to its destination. Tasks may be completed in an

arbitrary order. Furthermore, the crane should stand still for a unit duration at pick up and drop off.

The stacker cranes are identical and all share a single one dimensional track which lacks curvature, i.e. $f_i : [0, 1] \to \mathbb{R}$ and $f''(s) = 0$. Since the systems are one dimensional there is no use for a path function. Therefore we proceed using the state x_i rather than the path position s_i. The dynamics are defined by a double integrator

$$\ddot{x}_i(t) = u_i(t), \tag{7.1}$$

where u_i is the control input. Velocity and acceleration and bounded by

$$|\dot{x}_i(t)| \leq v^{\max}, \tag{7.2}$$
$$|\ddot{x}_i(t)| \leq a^{\max}, \tag{7.3}$$

where $v^{\max} \in \mathbb{R}^+$ and $a^{\max} \in \mathbb{R}^+$ are symmetric velocity and acceleration bounds, identical for all cranes. We note that the velocity state can now be negative. Finally there is an initial position and zero velocity boundary conditions

$$x_i(0) = x_i^0 \qquad \dot{x}_i(0) = \dot{x}_i(T) = 0, \tag{7.4}$$

where $x_i^0 \in \mathbb{R}$, $i \in \mathcal{N}$, is the crane initial position, and all systems share the same final time T.

For the collision avoidance constraints, suppose the cranes are placed along the track in order of index, and must keep a unit distance to avoid collisions. The latter without loss of generality. The collision avoidance constraint for two neighboring cranes i and $i + 1$ can then be posed as

$$x_i(t) + 1 \leq x_{i+1}(t). \tag{7.5}$$

Next are the processing and non-preemption conditions. Begin by defining a new vector of decision variables, $\mathbf{t} \in \mathbb{R}^{2m}$, describing the time instances at which each of the $2m$ locations are visited. Using the operators defined at the start of this section, the pair t_j^+ and t_j^-, describe the time of pickup and drop off for each task $j \in \mathcal{M}$. The processing constraint, i.e. each pick up and drop off location should be visited, for $j \in \mathcal{M}_2$, is posed as

$$x_i(t) = p_j \quad \forall t \in [t_j, t_j + 1] : \;\; i = a_j \tag{7.6}$$

which includes the unit time stand still condition. Note that the lazy modulo notation acts on a_j such that $a_j = a_{j+m}$. As for the non-preemption condition, for each pair of pick up and drop off, we have

$$(a_j = a_k) \rightarrow \left(t_j^- \leq t_k^+ \vee t_k^- \leq t_j^+\right), \tag{7.7}$$

i.e. the time intervals where tasks are processed may not overlap. We note that this looks very much like the scheduling disjunctions arising from collision zones, although this constraint describes the internal order of tasks on a single machine. Also, since an task must be picked up before it can be dropped off

$$t_j^+ \leq t_j^-. \tag{7.8}$$

The stacker crane problem can be summarized as

$$\begin{aligned}
&\min\ T \\
&\text{subject to} \\
&\left.\begin{array}{l} x_i(0) = x_i^0 \\ \dot{x}_i(0) = 0 \quad \dot{x}_i(T) = 0 \end{array}\right\} && \forall i \in \mathcal{N}, \\
&\left.\begin{array}{l} \ddot{x}_i(t) = u_i(t) \\ |\dot{x}_i(t)| \leq v^{\max} \\ |\ddot{x}_i(t)| \leq a^{\max} \end{array}\right\} && \forall t \in [0,T] \;\; i \in \mathcal{N}, \\
&\left.\begin{array}{l} t_j^+ \leq t_j^- \\ t_j^- + 1 \leq T \end{array}\right\} && \forall j \in \mathcal{M}, \\
&(a_j = a_k) \rightarrow \left(t_j^- \leq t_k^+ \vee t_k^- \leq t_j^+\right) && \forall j,k \in \mathcal{M} : \;\; j < k, \\
&x_i(t) = p_j \quad \forall t \in [t_j, t_j + 1] : \;\; i = a_j && \forall j \in \mathcal{M}_2,
\end{aligned} \tag{7.9}$$

where the decision variables are the assignment a_j, pick up and drop off times t_j, the state $x_i(t)$ and the final time T.

7.2 State space discretization

Consider the collision constraints in (7.5), if we could discretize our problem in time, these would become simple linear inequalities. Unfortunately, posing the processing constraints (7.6) becomes much too difficult, since the exact time at which pick up and drop off occurs is unknown.

In a state space discretization approach on the other hand, which we have focused on so far in this thesis, the collision avoidance constraint become problematic since we no longer have a fixed paths. But in the case of the time minimal problem, this difficulty may be overcome. We mentioned previously that a model presented in [96] included two cranes with unbounded acceleration occupying the same track. The main idea is that it is possible to ensure a collision free system by posing timing constraints on the visited positions. More specifically, we will keep the time instances as decision variables and try make due without explicitly describing the state. When the time instances are considered pairwise, we end up with two cases, either both

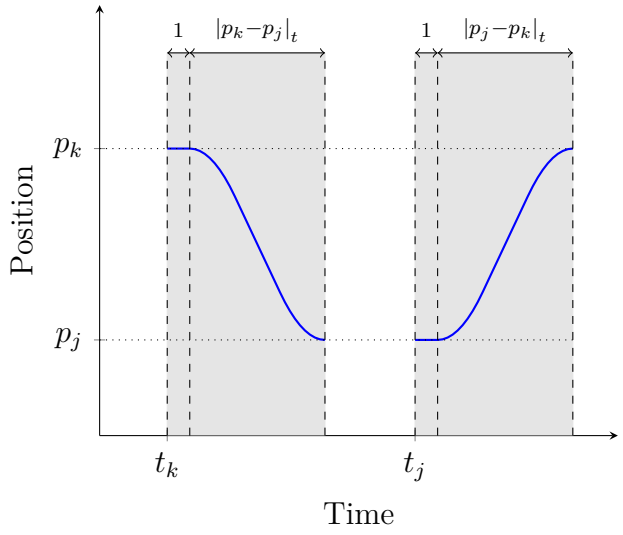

Figure 7.1: The blue lines show the projected minimum time solutions of moving from p_k to p_j and vise versa. There must be enough time to travel between locations according to (7.10), this can be regarded as a no-overlap between the grayed areas. In this figure $p_j = 6$ and $p_k = 2$.

elements of the pair are processed on the same machine, or on different machines. We will begin with the former case, which is also the simplest.

Consider two locations t_j and t_k processed on the same machine, if we can compute the minimum travel time between the locations, then the following constraint will ensure the state dynamics are fulfilled

$$\begin{aligned}(a_j = a_k) \rightarrow &\left(t_j^- + 1 + \left|p_j^- - p_k^+\right|_t \leq t_k^+\right) \vee \\ &\left(t_k^- + 1 + \left|p_k^- - p_j^+\right|_t \leq t_j^+\right) \qquad \forall j,k \in \mathcal{M} : \;\; j < k,\end{aligned} \tag{7.10}$$

where $|\cdot|_t$ denotes the minimum time to cover a given distance, assuming zero velocity boundary conditions, and $j < k$ is used to decrease redundancy, such that the constraint is only posed for unique pairs. In other words, if two locations j and k are visited by the same machine, the times must be separated by the travel time and the unit stand still condition. The constraint is illustrated in Figure 7.1, in which the two gray areas may not overlap in time. As for the case when both time instance belong to the same task, i.e. $j = k$, we pose

$$t_j^+ + 1 + \left|p_j^+ - p_j^-\right|_t \leq t_j^- \qquad \forall j \in \mathcal{M}, \tag{7.11}$$

which also makes (7.8) redundant.

Now, for the slightly more complicated case where location j is visited by a crane located higher than that of the crane serving location k, i.e. $a_j > a_k$, and the position p_j is at the same height or below the position of p_k. Note that if there are any cranes in between a_j and a_k, we must account for these as well. The collision avoidance constraint is then

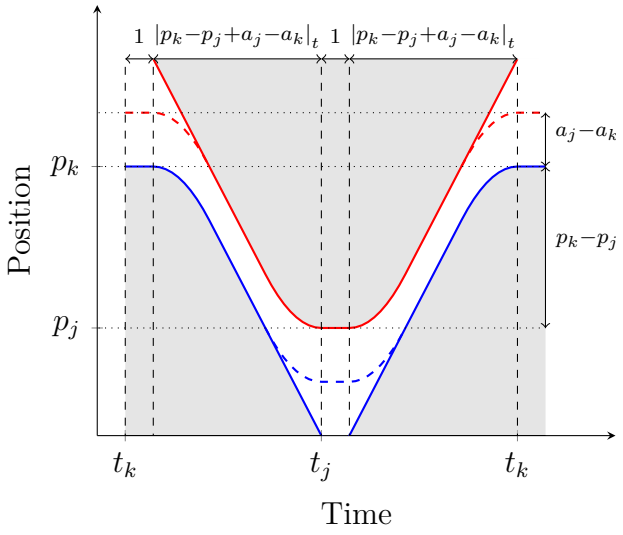

Figure 7.2: Two cranes must be separated by $a_j - a_k \geq 1$, crane a_j (red) travels into crane a_k:s (blue) territory below p_k in order to reach p_j. The time difference $|t_j - t_k|$ must always be larger or equal $1 + |p_k - p_j + a_j - a_k|_t$.

$$\begin{aligned}(a_j > a_k) \wedge (p_j < p_k + a_j - a_k) \rightarrow \\ \left(t_j + 1 + |p_k - p_j + a_j - a_k|_t \leq t_k\right) \vee \\ \left(t_k + 1 + |p_k - p_j + a_j - a_k|_t \leq t_j\right) \\ \forall j, k \in \mathcal{M}_2,\end{aligned} \tag{7.12}$$

where $p_k - p_j + a_j - a_k$ is the distance which must be cleared by crane a_j before and after it enters the area below p_k. We could consider this as crane a_j intruding into crane a_k's space at time t_j, and the timing constraint expresses the time it takes for crane a_j to clear the area. This constraint is illustrated in Figure 7.2. Since each crane is identical, this constraint can be regarded as ensuring there is enough time for a crane to accelerate and follow the other crane at unit distance.

Finally, the initial conditions are simply a special case of (7.10), where one of the time instances is zero, no unit wait time is imposed, and only pick up times are considered, i.e.

$$(a_j = i) \rightarrow \left(\left|p_j - x_i^0\right|_t \leq t_j^+\right) \quad \forall j \in \mathcal{M} \;\; i \in \mathcal{N}. \tag{7.13}$$

The space formulation can now be summarized as

$$\min \max_{j \in \mathcal{M}} \left(t_j^- + 1\right)$$

subject to

$$\begin{aligned}&(a_j = i) \rightarrow \left(|p_j - x_i^0|_t \leq t_j^+\right) && \forall j \in \mathcal{M} \;\; i \in \mathcal{N}, \\ &t_j^+ + 1 + \left|p_j^+ - p_j^-\right|_t \leq t_j^- && \forall j \in \mathcal{M},\end{aligned}$$

$$
\begin{aligned}
&(a_j = a_k) \rightarrow \\
&\quad \left(t_j^- + 1 + \left|p_k^- - p_j^+\right|_t \leq t_k^+\right) \vee \qquad \forall j,k \in \mathcal{M} : \;\; j < k, \\
&\quad \left(t_k^- + 1 + \left|p_k^- - p_j^+\right|_t \leq t_j^+\right) \\
&(a_j > a_k) \wedge (p_j < p_k + a_j - a_k) \rightarrow \\
&\quad \left(t_j + 1 + |p_k - p_j + a_j - a_k|_t \leq t_k\right) \vee \quad \forall j,k \in \mathcal{M}_2, \\
&\quad \left(t_k + 1 + |p_k - p_j + a_j - a_k|_t \leq t_j\right)
\end{aligned}
\tag{7.14}
$$

where the decision variables are now t_j and a_j, and the final time T has been substituted with a max function.

An easy way to simplify the problem is to assume that once a crane performs a pick up, it will move a minimum time speed to the drop off, i.e. using (7.10), any two time variables belonging to the same task, t_j^+ and t_j^-, are now

$$
t_j^+ + \left|p_j^+ - p_j^-\right|_t + 1 = t_j^-, \tag{7.15}
$$

which is an equality since we assumed that the crane moves in minimum time fashion. We can now substitute t_j^- into (7.10) and (7.12).

Furthermore, since each task is now performed in a time minimal fashion, we can further simplify (7.12) without affecting the optimal solution by much. Rather than to pose the constraints for $\forall j,k \in \mathcal{M}_2$, we can assume that if the pick up time of task k precedes any part of task j, then so will the drop off time. That is, we can reduce the constraint to act for $\forall j,k \in \mathcal{M}$, cutting the number of constraints down to a fourth. In our benchmark, we will see how these assumptions affects the computation time and quality of the solution. We will refer to this as the simplified model. Note that in Paper 6, this latter simplification was imposed for all models.

7.2.1 Constraint programming model

For the constraint programming model, we define a new set of variables $b \in \mathbb{Z}_2^{2m \times 2m}$, where each element b_{jk} describes the partial order of time instances where locations j and k are visited, i.e. $b_{jk} = 1$ if j precedes k and 0 otherwise. It follows that $b_{jk} = \neg b_{kj}$, for $j \neq k$. These variables are included to be used for branching decisions in the model. We will use the lazy modulo notation also for matrices, i.e. $b_{(j+m)(k+m)} = b_{jk}$.

Since p_j and p_k are constant, (7.10) and (7.12) can be aggregated into a constraint on the form

$$
\left(t_j + \Delta_{jk}\left(a_j - a_k\right) \leq t_k\right) \vee \left(t_k + \Delta_{kj}\left(a_j - a_k\right) \leq t_j\right) \quad \forall j,k \in \mathcal{M}_2, \tag{7.16}
$$

where Δ_{jk} and $\Delta_{kj} : \{(n-1), \ldots, (n-1)\} \rightarrow \mathbb{R}$ and describe the minimum allowed time-distance of t_j and t_k as a function of crane allocation. We note that Δ_{jk} may take a large negative value whenever it is not needed for collision avoidance. The initial conditions can be posed similarly

$$\left|p_j - \Delta^0(a_j)\right|_t \leq t_j^+ \quad \forall j \in \mathcal{M}, \tag{7.17}$$

where $\Delta^0(a_j) = x_i^0 : i = a_j$. Constraints on this form can be directly used in a constraint program with use of the element constraint. An element constraint is in essence a look-up-table, i.e. we can impose constraints on the from $y = r_k$, where k and y are variable, and $\mathbf{r}$ is a constant or variable vector.

Since we decided to base our model on the partial order of tasks, we shall reformulate (7.16). First, a large portion may be posed as

$$b_{jk} \rightarrow (t_j + \Delta_{jk}\,(a_j - a_k) \leq t_k) \quad \forall j,k \in \mathcal{M}_2 : \;\; j \neq k, \tag{7.18}$$

where $\Delta_{jk}\,(a_j - a_k)$ is defined as in (7.16). However, this excludes the case when $j = k$, i.e. when both indices belong to the same task. This is posed as

$$t_j^+ + 1 + \left|p_j^+ - p_j^-\right|_t \leq t_j^- \quad \forall j \in \mathcal{M}. \tag{7.19}$$

Furthermore, to model the non-preemption condition acting on tasks processed by the same crane, we add

$$a_j = a_k \rightarrow (b_{jk} = b_{j+n,k}) \wedge (b_{kj} = b_{k+n,j}) \quad \forall j,k \in \mathcal{M} \;\; j \neq k. \tag{7.20}$$

The constraint programming model can now be summarized as

$$\begin{aligned}
&\min \max_{j \in \mathcal{M}} \; \left(t_j^- + 1\right) \\
&\text{subject to} \\
&\quad |p_j - \Delta^0(a_j)|_t \leq t_j^+ && \forall j \in \mathcal{M}, \\
&\quad t_j^+ + 1 + |p_j - p_k|_t \leq t_j^- && \forall j \in \mathcal{M}, \\
&\quad b_{jk} \rightarrow (t_j + \Delta_{jk}\,(a_j - a_k) \leq t_k) && \forall j,k \in \mathcal{M}_2 : \;\; j \neq k, \\
&\quad a_j = a_k \rightarrow (b_{jk} = b_{j+n,k}) \wedge (b_{kj} = b_{k+n,j}) && \forall j,k \in \mathcal{M} : \;\; j \neq k, \\
&\quad b_{jk} = \neg b_{kj} && \forall j,k \in \mathcal{M}_2 : \;\; j \neq k,
\end{aligned} \tag{7.21}$$

where the decision variables are b_{jk}, a_j, t_j. In the case of the simplified model, $b \in \mathbb{Z}_2^{m \times m}$, and subsequently, the collision avoidance constraint (7.18) is only posed for all $j,k \in \mathcal{M}$ using appropriately modified Δ_{jk} functions. Furthermore, the same task constraint (7.19) becomes an equality and non-preemption (7.20) is simply removed.

7.2.2 Mathematical programming model

In a mixed integer linear programming model, we cannot be as expressive as in the constraint programming case. We will have to define binary variables to define the various configurations of our constraints and then pose all variants explicitly. First, we convert the integer vector of allocation variables $\mathbf{a}$ into a binary representation $d \in \mathbb{Z}_2^{n \times m}$, where each element d_{ij} describes whether or not crane i is assigned to task j. Next $s \in \mathbb{Z}_2^{m \times m}$ describes whether tasks are assigned to the same crane or not, $s_{jk} = 1$ if j and k are allocated to the same crane. Finally, $b \in \mathbb{Z}_2^{2m \times 2m}$ is the partial order of tasks, i.e. $b_{jk} = 1$ if j precedes k.

First of all, each task must be assigned to a crane

$$\sum_{i \in \mathcal{N}} d_{ij} = 1 \quad \forall j \in \mathcal{M}. \tag{7.22}$$

The relationship between assignment and the variable describing whether two tasks are allocated to the same crane must be defined

$$d_{ij} + d_{ik} - s_{jk} \leq 1 \quad \forall i \in \mathcal{N} \;\; j, k \in \mathcal{M}. \tag{7.23}$$

The equation for locations belonging to the same task (7.11) remains unchanged. Tasks on the same crane, described by (7.10), are converted into mixed integer constraints

$$t_j^- + 1 + \left| p_j^- - p_k^+ \right|_t \leq t_k^+ + M\,(2 - s_{jk} - b_{jk}) \quad \forall j, k \in \mathcal{M} : \;\; j \neq k, \tag{7.24}$$

where M is a sufficiently large constant. As for the initial position, the mixed integer version is

$$\left| p_j - x_i^0 \right|_t \leq t_j^+ + M\,(1 - d_{ij}) \quad \forall i \in \mathcal{N} \;\; j \in \mathcal{M}. \tag{7.25}$$

Finally, there is the collision avoidance constraint (7.12), which on mixed integer from is

$$\left.\begin{array}{l} t_j + 1 + |p_j - p_k + i - \ell|_t \leq t_k + M\,(3 - b_{jk} - d_{ij} - d_{\ell k}) \\ t_k + 1 + |p_j - p_k + i - \ell|_t \leq t_j + M\,(3 - b_{kj} - d_{ij} - d_{\ell k}) \end{array}\right\} \\ \forall i, \ell \in \mathcal{N} \;\; j, k \in \mathcal{M}_2 : \;\; (j \neq k) \quad (i > \ell) \quad (p_j < p_k + i - \ell). \tag{7.26}$$

where $j \neq k$ also includes $j \neq k + n$. We note that the constraint is posed for all combinations of both possible crane allocation and combination of visited locations.

The mixed integer linear formulation can now be summarized as

$$
\begin{aligned}
&\min \max_{j \in \mathcal{M}} \left(t_j^- + 1 \right) \\
&\text{subject to} \\
&\quad \textstyle\sum_{i \in \mathcal{N}} d_{ij} = 1 && \forall j \in \mathcal{M}, \\
&\quad d_{ij} + d_{ik} - s_{jk} \leq 1 && \forall i \in \mathcal{N} \;\; j,k \in \mathcal{M}, \\
&\quad |p_j - x_i^0|_t \leq t_j^+ + M\,(1 - d_{ij}) && \forall i \in \mathcal{N} \;\; j \in \mathcal{M}, \\
&\quad t_j^+ + 1 + \left| p_j^+ - p_j^- \right|_t \leq t_j^- && \forall j \in \mathcal{M}, \\
&\quad t_j^- + 1 + \left| p_j^- - p_k^+ \right|_t \leq t_k^+ + M\,(2 - s_{jk} - b_{jk}) && \forall j,k \in \mathcal{M} : \;\; j \neq k, \\
&\quad \left.\begin{aligned} t_j + 1 + |p_j - p_k + i - \ell|_t \leq t_k + M\,(3 - b_{jk} - d_{ij} - d_{\ell k}) \\ t_k + 1 + |p_j - p_k + i - \ell|_t \leq t_j + M\,(3 - b_{kj} - d_{ij} - d_{\ell k}) \end{aligned}\right\} \\
&\qquad \forall i,\ell \in \mathcal{N} \;\; j,k \in \mathcal{M}_2 : \;\; (j \neq k) \quad (i > \ell) \quad (p_j < p_k + i - \ell),
\end{aligned}
\tag{7.27}
$$

where the decision variables are now t_j, a_j, b_{jk}, s_{jk} and d_{ij}. In the case of the simplified model, $b \in \mathbb{Z}_2^{m \times m}$, and subsequently, an appropriately modified version of the collision avoidance constraint (7.26) is posed only for all $j,k \in \mathcal{M}$. Also, the same task constraint (7.11) becomes an equality.

7.3 Benchmark

Figure 7.3 shows a computational comparison between the constraint programming and mixed integer models. Each data point is the average solution time of 100 problem instances. The former was solved using Gecode and the latter using CPLEX. We can see that up till 10 tasks, Gecode performs better, whereafter CPLEX maintains a slight edge. We note that Gecode very much benefits from the proposed simplification.

Note that CPLEX was run through the AMPL modeling interface. It is possible that this introduces some overhead which could be avoided using an API. How much this affects CPLEX performance at a low number of tasks is unknown.

The proposed simplification in (7.15) will at times reduce the quality of the optimal solution. Since the absolute value of the optimal solution varies greatly with problem instance, we focus on the relative gap to evaluate the simplification. The gap between the simplified and exact models was non-zero in 53% of instances. For the non-zero instances, the average gap was 1.3% and the maximum gap 7.4%.

7.3.1 Local search

We have also implemented a local search algorithm which works in two phases, a constructive phase and a search phase. The goal of the constructive phase is to generate an initial solution. This is performed as follows: begin with a solution without any tasks, then add tasks one at a time, until all the tasks of the problem formulation has been introduced. Each time a task is added to the solution, a small

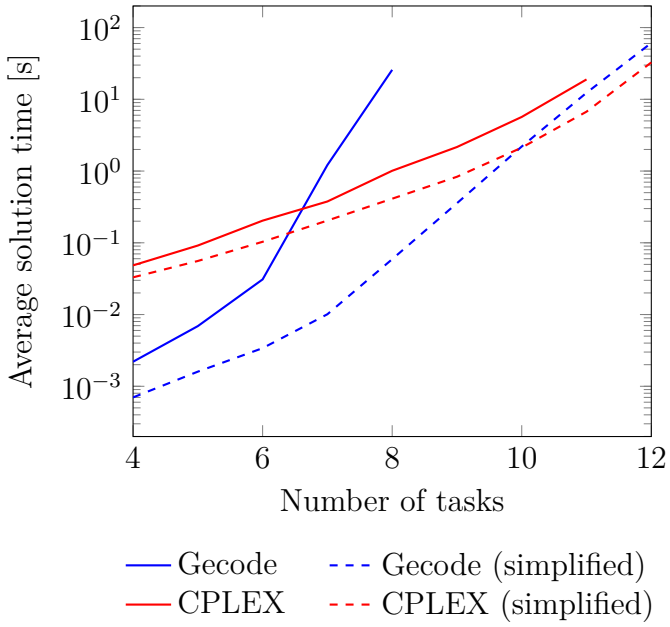

Figure 7.3: Average solution time for 100 randomly generated instances.

optimization problem is solved, sorting it amongst any existing tasks already added to the solution.

Next, during the search phase, a random set of tasks are selected iteratively. For each set of tasks, an optimization problem where the selected set of tasks should be resequenced, with regard to all other tasks is created. The tasks not in the set keep their partial order. That is, suppose there is an existing solution defined by the partial order **b** and assignment **a**, then rescheduling task k entails searching for the optimal value of row and column k in **b**, and element k of **a**.

The number of tasks in the randomly selected set may be varied. Figure 7.4 illustrates how this affects the results for a specific problem instance where the optimal solution is known up to 20 tasks. A time limit of $2m$ seconds was used to terminate the search. Clearly, near optimal solutions can be found in short times for up to at least 20 tasks.

The local search can be run for even 100 tasks. Unfortunately, the optimum of any problem instance over 20 tasks is unknown to us. Still, we can look at the convergence and determine how much time is needed to get a good solution. Figure 7.5 shows the results for such a 100 task instance. We can see that after 300 seconds, improvement in the local search is negligible.

7.4 Summary

In this chapter, we presented a state space discretization based method for a stacker crane scheduling problem. The state is only discretized at the pick up and drop off points, and collisions are implicitly handled using timing constraints. Using MILP or constraint programming, the problem can be solved reasonably fast for up to 12

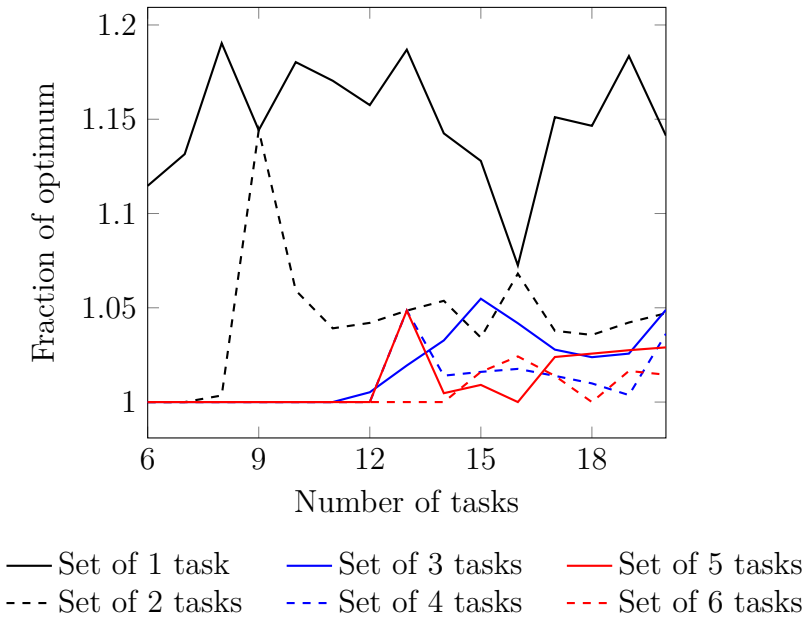

Figure 7.4: Local search result after 10 seconds as fraction of optimum solution.

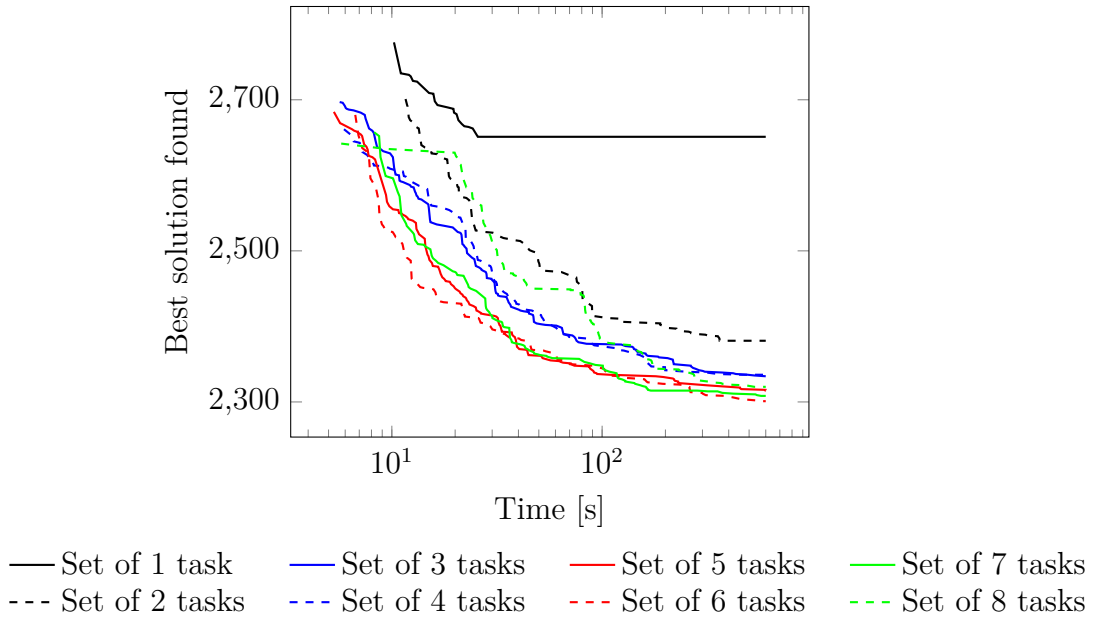

Figure 7.5: Local search result for instance of 100 tasks.

orders. Close to optimal solutions may be generated using local search, and we have verified good results up to 20 tasks. The local search can also find, what we believe are good, solutions for 100 task problem instances.

Chapter 8

Summary of appended papers

This chapter summarizes the papers contained in Part II. Important contributions are highlighted and the relation between papers is briefly discussed.

Paper 1

Oskar Wigström, Nikolce Murgovski, Sarmad Riazi and Bengt Lennartson. *Computationally efficient energy optimization of multiple robots.* Submitted for possible journal publication (under review), 2016.

This paper is focused on modeling the multi robot coordination problem. We pose the problem in the space domain and apply two different variable transformations found in the literature. Next, we propose criteria for energy minimization and present a case study where we study the computational efficiency of the models and energy measurements from an industrial robot.

Paper 2

Oskar Wigström, Bengt Lennartson, Alberto Vergnano and Claes Breitholtz *High-Level Scheduling of Energy Optimal Trajectories.* IEEE Transactions on Automation Science and Engineering, 10 (1), 57-64, 2013.

In this paper, we extend an existing approach to multi robot energy minimization found in [12]. The approach is based on paramterization of operation energy cost as a function of execution time, and the scheduling of these. In this paper we introduce a more energy efficient paramterization based on a dynamic programming.

Paper 3

Oskar Wigström, Bengt Lennartson and Sarmad Riazi. *Constraint programming and nonlinear scheduling.* Submitted for possible journal publication, 2016.

In this paper, we focus on the use of constraint propagation for nonlinear scheduling problems, in particular two problem classes found in literature. These correspond to the models resulting from Paper 1 and 2. We discuss existing methods and how constraint propagation can be used within these. An extensive benchmark is presented to demonstrate the efficiency of the method.

Paper 4

Sarmad Riazi, Oskar Wigström, Kristofer Bengtsson and Bengt Lennartson *Energy and Peak-power Optimization of Existing Time-optimal Robot Trajectories.* Under review for possible journal publication, 2016.

Here, we attempt to quantify the energy reduction available via multi robot scheduling. Varying load, peak power and extension of execution time is considered. Practical aspects are discussed.

Paper 5

Oskar Wigström and Bengt Lennartson *An integrated CP/OR method for optimal control of modular hybrid systems* IFAC Workshop on Discrete Event Systems, 47 (2), 485-491, 2014.

In this paper, we consider open loop control of synchronization of multi stage systems. Collocation is used for the continuous state dynamics and the integrated approach in Paper 3 is used to solve a small example.

Paper 6

Fredrik Hagebring, Oskar Wigström, Bengt Lennartson, Simon Ian Ware and Rong Su *Comparing MILP, CP, and A* for Multiple Stacker Crane Scheduling.* International Workshop on Discrete Event Systems, 2016.

This paper concerns the stacker crane scheduling problem. We extend an existing collision avoidance formulation to fit our problem. The resulting collision avoidance constraint is used to pose mixed integer linear programming, constraint programming and graph based models. These methods are then compared. A convenient simplification to the problem is also introduced.

Chapter 9

Conclusions and future work

At the beginning of Chapter 1, we posed three questions, or topics, which we have addressed throughout this thesis. These topics related to our problem formulation by: the level of detail encoded into a mathematical program; improvement of search algorithms by exploiting structure; and quantification of potential energy reduction. This chapter will attempt to conclude these topics and discuss directions of future work.

There are four categories of models presented in this thesis: the monolithic models in Section 3.1; the decomposition model in Section 3.2; the hybrid model in Chapter 6; and finally, the stacker crane model in Chapter 7. We will begin discussing the two former which are specifically used for energy efficient multi robot coordination.

Both approaches include velocity and acceleration constraints, and if model parameters are available, can also be extended to include torque constraints. The monolithic approach allows for a much higher resolution in synchronization than that of the decomposition model, since the practical implementation of the latter, requires robots to reach zero velocity at zone boundaries. The monolithic models are also computationally efficient, and in a fixed sequence setting these are highly preferable. However, the two monolithic models each have drawbacks. The kinetic energy model still includes non-convex constraints, and will suffer performance issues in the case that the sequence is unknown, and convex mixed integer algorithms cannot be used. Although, this is not a problem from the inverse velocity model, its inner approximation of the acceleration bounds may instead become problematic. The choice of model is clearly problem specific.

Regarding hybrid systems modeling, we conclude that while we can model more general dynamics, numerical issues tend to increase. This is due to the non-convexity in the approximation of the continuous dynamics, as well as the encoding of discrete dynamics into mixed integer constraints.

As for the stacker crane problem. It seems as if the solution space tends to become quite flat when the number of tasks increase. And thus, running a receding horizon scheme using the proposed models should yield quite good results.

Our approach to algorithmic improvement is to reduce the discrete search space during branch and bound. This is achieved by running constraint propagation methods each branch and bound search node, before other traditional mathematical

programming methods are run. In a benchmark, our methods outperforms other state of the art MINLP solvers for a range of problem instances. Even solvers which to some extent uses constraint propagation are surpassed. The level of speed-up is related to the computational complexity of the subproblems solved, as well as the number of feasible solutions. The more complex the subproblem, the more time there is to gain from decreasing the discrete search space. The less feasible solutions there are, the higher the likelihood of reducing the discrete search space.

The model resulting from the decomposition approach is the least computationally complex, and resembles a bounded time job shop problem with an added nonlinear cost. Although near time minimal final times resulted in large speed-ups, when the final time was extended and the number of feasible solutions increased, the constraint propagation methods became decreasingly cost efficient. In the case of the monolithic problem on the other hand, spending more time on constraint propagation methods is beneficial with increased problem size. That is, the most efficient approach is to use constraint propagation to search for full integer solutions, and solve the NLP subproblem only for these scheduling feasible solutions. For problems such as the hybrid model, which is even more complex, using tailored search methods to identify feasible solutions may prove a good approach.

As for the quantification of energy reduction, let us begin with a short note on the validity of our comparisons. Since we do not use a torque model, it becomes crucial to constrain velocity and acceleration such that robot's envelope of operation is not violated. This process may remove the true optimal solution from the feasible space. At the same time, some of the velocity and acceleration bounds imposed on the original robot trajectory are unnecessarily restrictive, and may be favorably relaxed in our model. These two factors may skew the results somewhat either against or in favor of our methods, respectively. However, this issue is mostly problematic at near time minimal solution, and most noticeable for the single robot problem.

From an energy reduction perspective, one can regard our approach as follows. In a system of multiple robots, there is typically some slack time in the system, i.e. the robots are not utilized fully at all times. The amount and distribution of available slack may vary depending on the sequence of tasks. Our algorithms search the sequence space for slack time configurations and allocate the slack amongst the robots in an optimal fashion.

From studying the single robot case, we have observed that the most significant energy reductions are achieved by moving away from time minimal solutions. If we disregard the effect of shared zones, as well as variations in payload and path, it seems reasonable that slack will be spread somewhat evenly amongst robots. This implies, in a best case scenario, the total energy reduction for all robots will proportionally mimic the reduction of the single robot. From our experiments with multiple robots, we have observed that given a sufficient slack time, $20 - 30\%$ energy may be saved.

Future work

Future work on energy efficient multi-robot coordination would be directed towards decentralization and robustness. In their current format, our methods are used

at a planning stage. The result is generated outside the robot and then encoded into the robot program. This makes reconfiguration on-site a problem. To create a more practical approach, our methods should be distributed into the robots. The sequencing problem may still have to be solved in a supervisory form, but the subproblems should be distributable.

Each individual system is governed by fixed deadlines which must be met, such as the total time. And pairwise, the systems are subject to collision avoidance timings constraints. A practical approach should enable the systems to pairwise negotiate these timing constraints. Some form of robustness which considers system breakdown must be included. Also disturbances in robot task execution time should be considered such that the fixed deadlines are always met.

Finally, a note on the stacker crane problem. The collision avoidance constraints look very much like a mutual exclusion, although the range of overlap is dependent on the order. It would be interesting to apply a sequence dependent no-overlap constraint.

Bibliography

[1] M. Todtermuschke, M. Findeisen, and A. Bauer, "Methodology for Creation a Reference Trajectory for Energetic Comparability of Industrial Robots in Body Shop," *Procedia CIRP*, vol. 23, pp. 122–126, 2014, ISSN: 2212-8271. DOI: 10.1016/j.procir.2014.10.085.

[2] N. Wemhöner, "Flexibilitätsoptimierung zur Auslastungssteigerung im Automobilrohbau," PhD thesis, Aachen, 2006, XI, 207 S. : Ill., graph. Darst.

[3] D. Meike and L. Ribickis, "Energy efficient use of robotics in the automobile industry," in *Advanced Robotics (ICAR), 2011 15th International Conference on*, 2011, pp. 507–511. DOI: 10.1109/ICAR.2011.6088567.

[4] J. Engelmann, *Methoden und Werkzeuge zur Planung und Gestaltung energieeffizienter Fabriken*. IBF, 2009.

[5] E. Commision, "Energy Efficiency Plan," Brussels, Tech. Rep., 2011.

[6] O. Maimon, E. Profeta, and S. Singer, "Energy analysis of robot task motions," *Journal of Intelligent and Robotic Systems*, vol. 4, no. 2, pp. 175–198, 1991, ISSN: 0921-0296. DOI: 10.1007/BF00440418.

[7] F. Roos, H. Johansson, and J. Wikander, "Optimal selection of motor and gearhead in mechatronic applications," *Mechatronics*, vol. 16, no. 1, pp. 63–72, 2006, ISSN: 0957-4158. DOI: DOI:10.1016/j.mechatronics.2005.08.001.

[8] T. Izumi, H. Zhou, and Z. Li, "Optimal Design of Gear Ratios and Offset for Energy Conservation of an Articulated Manipulator," *Automation Science and Engineering, IEEE Transactions on*, vol. 6, no. 3, pp. 551–557, 2009, ISSN: 1545-5955. DOI: 10.1109/TASE.2008.2010880.

[9] Z. Shiller, "Time-energy optimal control of articulated systems with geometric path constraints," *Journal of dynamic systems, measurement, and control*, vol. 118, no. 1, pp. 139–143, 1994. DOI: 10.1109/ROBOT.1994.350931.

[10] O. Stryk and M. Schlemmer, "Computational Optimal Control," in, R. Bulirsch and D. Kraft, Eds. Basel: Birkh{ä}user Basel, 1994, ch. Optimal Co, pp. 367–382, ISBN: 978-3-0348-8497-6. DOI: 10.1007/978-3-0348-8497-6_30.

[11] A. Kobetski and M. Fabian, "Velocity balancing in flexible manufacturing systems," in *Discrete Event Systems, 2008. WODES 2008. 9th International Workshop on*, 2008, pp. 358–363. DOI: 10.1109/WODES.2008.4605972.

[12] A. Vergnano, C. Thorstensson, B. Lennartson, P. Falkman, M. Pellicciari, F. Leali, and S. Biller, "Modeling and Optimization of Energy Consumption in Cooperative Multi-Robot Systems," *Automation Science and Engineering, IEEE Transactions on*, vol. 9, no. 2, pp. 423–428, Apr. 2012, ISSN: 1545-5955. DOI: 10.1109/TASE.2011.2182509.

[13] Paryanto, M. Brossog, M. Bornschlegl, and J. Franke, "Reducing the energy consumption of industrial robots in manufacturing systems," *The International Journal of Advanced Manufacturing Technology*, pp. 1–14, 2015, ISSN: 0268-3768. DOI: 10.1007/s00170-014-6737-z.

[14] D. Meike, M. Pellicciari, and G. Berselli, "Energy Efficient Use of Multirobot Production Lines in the Automotive Industry: Detailed System Modeling and Optimization," *Automation Science and Engineering, IEEE Transactions on*, vol. 11, no. 3, pp. 798–809, 2014, ISSN: 1545-5955. DOI: 10.1109/TASE.2013.2285813.

[15] Z. Yan, N. Jouandeau, and A. A. Cherif, "A survey and analysis of multi-robot coordination," *International Journal of Advanced Robotic Systems*, vol. 10, 2013. DOI: 10.5772/57313.

[16] J. Peng and S. Akella, "Coordinating multiple robots with kinodynamic constraints along specified paths," *The International Journal of Robotics Research*, vol. 24, no. 4, pp. 295–310, 2005. DOI: 10.1007/978-3-540-45058-0_14.

[17] K. Dresner and P. Stone, "Multiagent traffic management: A reservation-based intersection control mechanism," in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*, IEEE Computer Society, 2004, pp. 530–537. DOI: 10.1613/jair.2502.

[18] N. Murgovski, G. R. de Campos, and J. Sjöberg, "Convex modeling of conflict resolution at traffic intersections," in *2015 54th IEEE Conference on Decision and Control (CDC)*, Dec. 2015, pp. 4708–4713. DOI: 10.1109/CDC.2015.7402953.

[19] M. R. Hafner, D. Cunningham, L. Caminiti, and D. Del Vecchio, "Cooperative collision avoidance at intersections: Algorithms and experiments," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 3, pp. 1162–1175, 2013. DOI: 10.1109/TITS.2013.2252901.

[20] O. Wigström and B. Lennartson, "Integrated OR/CP optimization for discrete event systems with nonlinear cost," in *Proceedings of the IEEE Conference on Decision and Control*, Institute of Electrical and Electronics Engineers Inc., 2013, pp. 7627–7633. DOI: 10.1109/CDC.2013.6761100.

[21] J. E. Bobrow, S. Dubowsky, and J. Gibson, "Time-Optimal Control of Robotic Manipulators Along Specified Paths," *IJRR*, vol. 4, no. 3, pp. 3–17, 1985. DOI: 10.1177/027836498500400301.

[22] P. A. O'Donnell and T. Lozano-Pérez, "Deadlock-free and collision-free coordination of two robot manipulators," in *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*, IEEE, 1989, pp. 484–489. DOI: 0.1109/ROBOT.1989.100033.

[23] T. Simeon, S. Leroy, and J. P. Lauumond, "Path coordination for multiple mobile robots: a resolution-complete algorithm," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 1, pp. 42–49, Feb. 2002, ISSN: 1042-296X. DOI: 10.1109/70.988973.

[24] S. M. LaValle and S. A. Hutchinson, "Optimal motion planning for multiple robots having independent goals," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 6, pp. 912–925, Dec. 1998, ISSN: 1042-296X. DOI: 10.1109/70.736775.

[25] L. Bruni, A. Colombo, and D. Del Vecchio, "Robust multi-agent collision avoidance through scheduling," 2013. DOI: 10.1109/CDC.2013.6760492.

[26] A. S. Manne, "On the Job-Shop Scheduling Problem," *Operations Research*, vol. 8, no. 2, pp. 219–223, 1960. DOI: 10.1287/opre.8.2.219.

[27] T. Siméon, S. Leroy, and J.-P. Laumond, "A collision checker for car-like robots coordination," in *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, IEEE, vol. 1, 1998, pp. 46–51. DOI: 10.1109/ROBOT.1998.676252.

[28] M. R. Shoaei, B. Lennartson, and S. Miremadi, "Automatic generation of controllers for collision-free flexible manufacturing systems," in *2010 IEEE International Conference on Automation Science and Engineering*, IEEE, 2010, pp. 368–373. DOI: 10.1109/COASE.2010.5584539.

[29] J. N. Hooker and M. A. Osorio, "Mixed Logical/Linear Programming," *Discrete Applied Mathematics*, vol. 96, pp. 96–97, 1997. DOI: 10.1016/S0166-218X(99)00100-6.

[30] P. Bonami, M. Kilinç, and J. Linderoth, "Algorithms and Software for Convex Mixed Integer Nonlinear Programs," in *Mixed Integer Nonlinear Programming*, ser. The IMA Volumes in Mathematics and its Applications, J. Lee and S. Leyffer, Eds., vol. 154, Springer New York, 2012, pp. 1–39. DOI: 10.1007/978-1-4614-1927-3_1.

[31] O. K. Gupta and A. Ravindran, "Branch and Bound Experiments in Convex Nonlinear Integer Programming," *Management Science*, vol. 31, no. 12, pp. 1533–1546, 1985. DOI: 10.1287/mnsc.31.12.1533.

[32] R. J. Dakin, "A tree-search algorithm for mixed integer programming problems," *The Computer Journal*, vol. 8, no. 3, pp. 250–255, 1965. DOI: 10.1093/comjnl/8.3.250.

[33] M. Duran and I. Grossmann, "An outer-approximation algorithm for a class of mixed-integer nonlinear programs," *Mathematical Programming*, vol. 36, no. 3, pp. 307–339, 1986. DOI: 10.1007/BF02592064.

[34] R. Fletcher and S. Leyffer, "Solving mixed integer nonlinear programs by outer approximation," *Mathematical Programming*, vol. 66, no. 1, pp. 327–349, 1994. DOI: 10.1007/BF01581153.

[35] I. Quesada and I. E. Grossmann, "An LP/NLP based branch and bound algorithm for convex MINLP optimization problems," *Computers and Chemical Engineering*, vol. 16, no. 10-11, pp. 937–947, 1992. DOI: 10.1016/0098-1354(92)80028-8.

[36] T. Westerlund and F. Pettersson, "An extended cutting plane method for solving convex MINLP problems," *Computers & Chemical Engineering*, vol. 19, pp. 131–136, 1995. DOI: 10.1016/0098-1354(95)87027-X.

[37] I. E. Grossmann and S. Lee, "Generalized Convex Disjunctive Programming: Nonlinear Convex Hull Relaxation," *Computational Optimization and Applications*, vol. 26, no. 1, pp. 83–100, 2003, ISSN: 0926-6003. DOI: 10.1023/A:1025154322278.

[38] F. Rossi, P. van Beek, and T. Walsh, *Handbook of Constraint Programming*, ser. Foundations of Artificial Intelligence. Elsevier Science, 2006, ISBN: 9780444527264.

[39] K. Apt, *Principles of Constraint Programming*. Cambridge University Press, 2003, ISBN: 0521825830.

[40] J. N. Hooker and G. Ottosson, "Logic-based Benders decomposition," *Mathematical Programming*, vol. 96, no. 1, pp. 33–60, 2003. DOI: 10.1007/s10107-003-0375-9.

[41] Gecode Team, *Gecode: Generic Constraint Development Environment*, 2016. [Online]. Available: http://www.gecode.org.

[42] P. Baptiste and C. L. Pape, "Edge-Finding Constraint Propagation Algorithms for Disjunctive and Cumulative Scheduling," in *Scheduling," Proceedings 15 th Workshop of the U.K. Planning Special Interest Group*, 1996. DOI: 10.1023/A:1009822502231.

[43] D. Verscheure, B. Demeulenaere, J. Swevers, J. D. Schutter, and M. Diehl, "Time-Optimal Path Tracking for Robots: A Convex Optimization Approach," *IEEE Transactions on Automatic Control*, vol. 54, no. 10, pp. 2318–2327, 2009, ISSN: 0018-9286. DOI: 10.1109/TAC.2009.2028959.

[44] K. Shin and N. McKay, "A dynamic programming approach to trajectory planning of robotic manipulators," *Automatic Control, IEEE Transactions on*, vol. 31, no. 6, pp. 491–500, 1986, ISSN: 0018-9286. DOI: 10.1109/TAC.1986.1104317.

[45] S. Riazi, K. Bengtsson, O. Wigstrom, E. Vidarsson, and B. Lennartson, "Energy optimization of multi-robot systems," in *2015 IEEE International Conference on Automation Science and Engineering (CASE)*, IEEE, 2015, pp. 1345–1350, ISBN: 978-1-4673-8183-3. DOI: 10.1109/CoASE.2015.7294285.

[46] T. Lipp and S. Boyd, "Minimum-time speed optimisation over a fixed path," *International Journal of Control*, vol. 87, no. 6, pp. 1297–1311, 2014. DOI: 10.1080/00207179.2013.875224.

[47] J. M. Hollerbach, "Dynamic scaling of manipulator trajectories," *Journal of Dynamic Systems, Measurement, and Control*, vol. 106, no. 1, pp. 102–106, 1984. DOI: 10.1115/1.3149652.

[48] R. Fourer, D. Gay, and B. Kernighan, *AMPL: a mathematical programming language*. Boyd & Fraser, 1993, vol. 117, ISBN: 0-534-38809-4.

[49] A. Wächter and B. T. Lorenz, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2005, ISSN: 1436-4646. DOI: 10.1007/s10107-004-0559-y.

[50] S. G. Robert Hult Mario Zanon and P. Falcone, "Primal Decomposition of the Optimal Coordination of Vehicles at Traffic Intersections," in *Proceedings of the IEEE Conference on Decision and Control*, 2016.

[51] G. Field and Y. Stepanenko, "Iterative dynamic programming: an approach to minimum energy trajectory planning for robotic manipulators," in *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, vol. 3, Apr. 1996, 2755–2760 vol.3. DOI: 10.1109/ROBOT.1996.506579.

[52] V. Jain and I. E. Grossmann, "Algorithms for Hybrid MILP/CP Models for a Class of Optimization Problems," *INFORMS J. on Computing*, vol. 13, no. 4, pp. 258–276, Sep. 2001, ISSN: 1526-5528. DOI: 10.1287/ijoc.13.4.258.9733.

[53] M. Lombardi and M. Milano, "Optimal methods for resource allocation and scheduling: a cross-disciplinary survey," *Constraints*, vol. 17, no. 1, pp. 51–85, 2012, ISSN: 1383-7133. DOI: 10.1007/s10601-011-9115-6.

[54] P. Van Hentenryck, *The OPL optimization programming language*. Mit Press, 1999, ISBN: 0-262-72030-2.

[55] T. Yunes, I. D. Aron, and J. N. Hooker, "An Integrated Solver for Optimization Problems," *Oper. Res.*, vol. 58, no. 2, pp. 342–356, 2010, ISSN: 0030-364X. DOI: 10.1287/opre.1090.0733.

[56] P. Laborie and J. Rogerie, "Temporal Linear Relaxation in IBM ILOG CP Optimizer," in *Proceedings of 6th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2013)*, Aug. 2013. DOI: 10.1007/s10951-014-0408-7.

[57] M. Tawarmalani and N. Sahinidis, "A polyhedral branch-and-cut approach to global optimization," *Mathematical Programming*, vol. 103, no. 2, pp. 225–249, 2005. DOI: 10.1007/s10107-005-0581-8.

[58] T. Berthold and K. Wolter, "Constraint Integer Programming: a New Approach to Integrate CP and MIP," 2008. DOI: 10.1007/978-3-540-68155-7_4.

[59] *Coin OR: Linear Programming solver CLP*. [Online]. Available: https://projects.coin-or.org/Clp.

[60] O. Wigström and B. Lennartson, "An Integrated CP/OR Method for Optimal Control of Modular Hybrid Systems," in *Proc. 12th IFAC-IEEE International Workshop on Discrete Event Systems (WODES'14), Cachan, France, May*, 2014, pp. 485–491, ISBN: 978-3-902823-61-8. DOI: 10.3182/20140514-3-FR-4046.00130.

[61] G. O. Inc., *Gurobi Optimizer Reference Manual*, 2015. [Online]. Available: http://www.gurobi.com.

[62] R. H. Byrd, J. Nocedal, and R. A. Waltz, "KNITRO: An integrated package for nonlinear optimization," in *Large-Scale Nonlinear Optimization*, G. di Pillo and M. Roma, Eds., Springer, 2006, pp. 35–59. DOI: 10.1007/0-387-30065-1_4.

[63] *Coin OR: Mixed Integer Linear Programming solver CBC*, 2007. [Online]. Available: http://www.coin-or.org/projects/Cbc.xml.

[64] M. R. Bussieck and S. Vigerske, "MINLP solver software," *Wiley encyclopedia of operations research and management science*, 2010. DOI: 10.1002/9780470400531.eorms0527.

[65] L. Sciavicco, B. Siciliano, and B. Sciavicco, *Modelling and Control of Robot Manipulators*, 2nd. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2000, pp. 131–140, ISBN: 1852332212. DOI: 10.1007/978-1-4471-0449-0.

[66] J. S. Park, "Motion profile planning of repetitive point-to-point control for maximum energy conversion efficiency under acceleration conditions," *Mechatronics*, vol. 6, no. 6, pp. 649–663, 1996, ISSN: 0957-4158. DOI: DOI:10.1016/0957-4158(96)00012-8.

[67] D. Y. Ohm. (). (2006, Feb. 3). Selection of servo motors and drives (rev.2): http://www.drivetechinc.com, [Online]. Available: http://www.drivetechinc.com/.

[68] P. I. Barton and C. K. Lee, "Modeling, simulation, sensitivity analysis, and optimization of hybrid systems," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 12, no. 4, pp. 256–289, 2002. DOI: 10.1145/643120.643122.

[69] M. S. Branicky, V. S. Borkar, and S. K. Mitter, "A unified framework for hybrid control: model and optimal control theory," *Automatic Control, IEEE Transactions on*, vol. 43, no. 1, pp. 31–45, Jan. 1998, ISSN: 0018-9286. DOI: 10.1109/9.654885.

[70] S. Hedlund and A. Rantzer, "Optimal control of hybrid systems," in *Decision and Control, 1999. Proceedings of the 38th IEEE Conference on*, IEEE, vol. 4, 1999, pp. 3972–3977. DOI: 10.1109/CDC.1999.827981.

[71] S. Hedlund and A. Rantzer, "Convex dynamic programming for hybrid systems," *Automatic Control, IEEE Transactions on*, vol. 47, no. 9, pp. 1536–1540, 2002. DOI: 10.1109/TAC.2002.802753.

[72] H. J. Sussmann, "A maximum principle for hybrid optimal control problems," in *Decision and Control, 1999. Proceedings of the 38th IEEE Conference on*, Ieee, vol. 1, 1999, pp. 425–430. DOI: 10.1109/CDC.1999.832814.

[73] B. Passenberg, P. E. Caines, M. Sobotka, O. Stursberg, and M. Buss, "The minimum principle for hybrid systems with partitioned state space and unspecified discrete state sequence," in *Decision and Control (CDC), 2010 49th IEEE Conference on*, Dec. 2010, pp. 6666–6673. DOI: 10.1109/CDC.2010.5717263.

[74] M. Shahid Shaikh and P. E. Caines, "On the Hybrid Optimal Control Problem: Theory and Algorithms," *Automatic Control, IEEE Transactions on*, vol. 52, no. 9, pp. 1587–1603, Sep. 2007, ISSN: 0018-9286. DOI: 10.1109/TAC.2007.904451.

[75] P. Riedinger, J. Daafouz, and C. Iung, "About solving hybrid optimal control problems," *IMACS05*, 2005.

[76] M. P. Avraam, N. Shah, and C. C. Pantelides, "Modelling and optimisation of general hybrid systems in the continuous time domain," *Computers & chemical engineering*, vol. 22, S221–S228, 1998. DOI: 10.1016/S0098-1354(98)00058-1.

[77] J. Oldenburg and W. Marquardt, "Disjunctive modeling for optimal control of hybrid systems," *Computers & chemical engineering*, vol. 32, no. 10, pp. 2346–2364, 2008. DOI: 10.1016/j.compchemeng.2007.12.002.

[78] D. Garg, "Advances in global pseudospectral methods for optimal control," PhD thesis, University of Florida, 2011.

[79] A. Bemporad and M. Morari, "Control of systems integrating logic, dynamics, and constraints," *Automatica*, vol. 35, no. 3, pp. 407–427, 1999. DOI: 10.1016/S0005-1098(98)00178-2.

[80] G. Ferrari-Trecate, F. A. Cuzzola, D. Mignone, and M. Morari, "Analysis of discrete-time piecewise affine and hybrid systems," *Automatica*, vol. 38, no. 12, pp. 2139–2146, 2002. DOI: 10.1016/S0005-1098(02)00142-5.

[81] M. Rubagotti, S. Trimboli, and A. Bemporad, "Stability and invariance analysis of uncertain discrete-time piecewise affine systems," *IEEE Transactions on Automatic Control*, vol. 58, no. 9, pp. 2359–2365, 2013. DOI: 10.1109/TAC.2013.2251774.

[82] A. Bemporad, F. Borrelli, and M. Morari, "Piecewise linear optimal controllers for hybrid systems," in *American Control Conference, 2000. Proceedings of the 2000*, IEEE, vol. 2, 2000, pp. 1190–1194. DOI: 10.1109/ACC.2000.876688.

[83] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho, "Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems," Springer-Verlag, 1993, pp. 209–229. DOI: 10.1007/3-540-57318-6_30.

[84] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. DOI: 10.1007/978-0-387-68612-7.

[85] B. Lennartson, F. Basile, S. Miremadi, Z. Fei, M. N. Hosseini, M. Fabian, and K. Åkesson, "Supervisory Control for State-Vector Transition Models - A Unified Approach," *IEEE Transaction on Automation Science and Engineering*, vol. 11, no. 1, 2014. DOI: 10.1109/TASE.2013.2291115.

[86] L. Ghomri and H. Alla, "Modeling and analysis using hybrid Petri nets," *Nonlinear Analysis: Hybrid Systems*, vol. 1, no. 2, pp. 141–153, 2007, ISSN: 1751-570X. DOI: 10.1016/j.nahs.2006.04.004.

[87] R. Champagnat, P. Esteban, H. Pingaud, and R. Valette, "Petri net based modeling of hybrid systems," *Computers in Industry*, vol. 36, no. 1-2, pp. 139–146, 1998, ISSN: 0166-3615. DOI: 10.1016/S0166-3615(97)00109-7.

[88] N. Lynch, R. Segala, and F. Vaandrager, "Hybrid I/O automata," *Information and Computation*, vol. 185, no. 1, pp. 105–157, 2003, ISSN: 0890-5401. DOI: 10.1016/S0890-5401(03)00067-1.

[89] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee, "Modular specification of hybrid systems in CHARON," in *Hybrid Systems: Computation and Control*, Springer, 2000, pp. 6–19. DOI: 10.1007/3-540-46430-1_5.

[90] 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, 2009, ISSN: 0924-6703. DOI: 10.1007/s10626-009-0064-9.

[91] C. Thorstensson, S. Kanthabhabhajeya, B. Lennartson, and P. Falkman, "Optimization of Discrete Event Systems Using Extended Finite Automata and Mixed-Integer Nonlinear Programming," in *Proceedings of the 18th IFAC World Congress, 2011*, vol. 18, 2011. DOI: 10.3182/20110828-6-IT-1002.03630.

[92] D. Benson, "A Gauss pseudospectral transcription for optimal control," PhD thesis, Massachusetts Institute of Technology, 2005.

[93] P. J. Davis, *Interpolation and approximation*. Dover publications, 1975, ISBN: 0486624951.

[94] P. Bonami, L. T. Biegler, A. R. Conn, G. Cornuéjols, I. E. Grossmann, C. D. Laird, J. Lee, A. Lodi, F. Margot, and N. Sawaya, "An algorithmic framework for convex mixed integer nonlinear programs," *Discrete Optimization*, vol. 5, no. 2, pp. 186–204, 2008. DOI: 10.1016/j.disopt.2006.10.011.

[95] F. Domes and A. Neumaier, "Constraint propagation on quadratic constraints," *Constraints*, vol. 15, no. 3, pp. 404–429, 2010. DOI: 10.1007/s10601-009-9076-1.

[96] G. Belov, N. Boland, M. W. P. Savelsbergh, and P. Stuckey, "Local Search for a Cargo Assembly Planning Problem," in *Integration of AI and OR Techniques in Constraint Programming*, ser. Lecture Notes in Computer Science, vol. 8451, 2014, pp. 159–175. DOI: 10.1007/978-3-319-07046-9_12.