Characterizing Performance Variability in Manufacturing System Configurations

Özbay, İzgi

Award date:
2022

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Electronic Systems Research Group

Characterizing Performance Variability in Manufacturing System Configurations

Master Thesis

İzgi Özbay

Supervisors:
Twan Basten (TU/e Supervisor)
Jacques Verriet (TNO Supervisor)
Bram van der Sanden (TNO Supervisor)

Eindhoven, August 2022

Contents

Contents **ii**

1 Introduction **1**
1.1 Performance Variability in FMS 1
1.2 Variability in FMS Critical Paths 2
1.3 LSAT 2
1.4 Contributions 3
1.5 Thesis Outline 3

2 Related Work **4**

3 Preliminaries **7**
3.1 Modeling Concepts of LSAT 7
3.2 Notations 10
3.3 Parametric Critical-Path Problem 11
3.4 Parameterised LSAT Model 11
3.5 Characterization Problem 12
3.6 Third-Order Point-To-Point Motion Profiles 13

4 Polytope-Based Characterization **14**
4.1 Polytope Algorithm Example 14
4.2 Polytope Algorithm Pseudocode 22
4.3 Conclusion 29

5 Hyperrectangle-Based Characterization **31**
5.1 Hyperrectangle Algorithm Example 31
5.2 Hyperrectangle Algorithm Pseudocode 35
5.3 Conclusion 38

6 Performance Analysis of the Characterization Algorithms **39**
6.1 Characteristics of the Algorithms 39
6.2 Experiments and Results 40
6.3 Conclusion 50

7 LSAT Extension **52**
7.1 Critical-Path Analysis of LSAT Models with Fixed Action Durations 52
7.2 Critical-Path Analysis of Parameterised LSAT Models 55

7.3 Conclusion 61

8 Conclusion and Future Work **63**
8.1 Conclusion 63
8.2 Future Work 64

Bibliography **65**

Chapter 1

Introduction

Increasing complexity and demand for flexibility have been important factors in the evolution of manufacturing systems throughout the last decades. Before the eighties, manufacturing equipment was mainly dominated by dedicated manufacturing systems. These dedicated systems were typically used for manufacturing a single product at a high production rate. This made it possible to achieve low cost per product [34]. With increasing need for mass customization and for greater responsiveness to changes in products, dedicated manufacturing systems could no longer satisfy the demands of markets. Flexible manufacturing was introduced as a response to such needs and flexible manufacturing systems (FMS) were developed to address mid-volume, mid-variety production needs [49].

1.1 Performance Variability in FMS

In order to address variable production needs, FMS need to be configured for different circumstances. Three main sources of variations are product variability, production process variability and machine resource variability (PPR variability) [38]. To obtain an effective FMS, designers need to optimize an FMS's performance under different conditions. With the variability and increased complexity that needs to be addressed, an FMS's design space is very large. Because of the sheer size of this design space, finding the best trade-off between performance characteristics, like throughput and makespan, and other system-level concerns, such as cost, is very challenging.

Variability in PPR implies that the performance of an FMS depends on many parameters, both numerical parameters and categorical ones. Together, these parameters with their possible values define the *parameter space* of the FMS. An important question is *how performance of the FMS depends on these parameters.*

Manual parameter-space exploration is time consuming and, because of the size of the space, it is almost impossible to explore the entire space. In order to handle the complexity, [45] advocates *model-driven* system-performance engineering (MD-SysPE). System-level models are critical in MD-SysPE, as they act as the single source of truth driving the entire performance analysis. Therefore, we need to create models at a high abstraction level that are accurate enough to explore the system performance during early system development.

1.2 Variability in FMS Critical Paths

For the performance analysis of manufacturing systems, critical-path analysis is especially significant considering that the critical path(s) of a system determine important performance characteristics such as makespan and throughput. With critical-path analysis, a system's performance bottlenecks are detected and the overall performance can be improved by alleviating these bottlenecks.

Compared to dedicated manufacturing systems, because of PPR variability, FMS behavior is characterized by different scenarios. Between these different scenarios, there are variations in the product, the used resources and the production process. Therefore, FMS bottlenecks also vary in between different scenarios. The overall performance of an FMS is determined by the combination of its performance in different scenarios. Then, the detection and alleviation of different bottlenecks in different scenarios play an important role for improving the overall performance of FMS.

1.3 LSAT

The Logistics Specification and Analysis Tool (LSAT) [44] is a tool for rapid design-space exploration of supervisory controllers in FMS. LSAT is based on domain-specific languages. In contrast to other languages that have a generic syntax to support a broad range of systems such as the ones used by POOSL [47], UPPAAL [36] and mCRL2 [21], LSAT is designed to specifically support performance analysis during the early design phase of FMS. This enables modeling at a higher level of abstraction without the need for encoding domain concepts. LSAT uses precise and explicit formal models of so-called activities that have sufficient detail to facilitate design-space exploration. It further exploits the structure of these models to prune the design space and improve scalability of the performance analysis. LSAT supports, among others, makespan analysis and critical-path analysis.

Currently, LSAT is being extended by parameterisation of its domain-specific languages. The goal of this extension is to achieve models that enable expression of PPR variability. With the development of proper performance analysis techniques capable of analysing such parameterised models, LSAT's current analysis capabilities, which are mostly limited to analyzing performance bounds of a concrete model, are going to be expanded with analysis of how performance depends on the choices for the parameters defining the parameter space. We define this problem as the *parameter-space performance characterization problem.*

The goal for characterization problems is to capture the impact of parameter values on the performance throughout the entire parameter space. Note that characterization is different from optimization. In contrast to characterization, the goal for optimization problems is to find specific point(s) in the parameter space that optimize a certain performance criterion. An example of an optimization problem is to find the best-case throughput of an FMS and the parameter values realizing this throughput, while the corresponding characterization problem aims to capture how the highest achievable throughput depends on the parameter values.

1.4 Contributions

The main focus of this thesis is the parameter-space characterization for the critical path(s) of a parameterised LSAT model with a given parameter space of interest. We define the corresponding problem as the *parametric critical-path problem*. This is an instance of the already mentioned *characterization problem*. With certain assumptions, the parametric critical-path problem can be solved by solving the corresponding characterization problem. The two main contributions of this thesis are given as the following:

- **Contribution 1:** Two different algorithms are designed and implemented to solve the characterization problem. One of them is a polytope-based characterization algorithm that does the characterization by partitioning the parameter space into polytopes. The other one is a hyperrectangle-based algorithm which partitions the parameter space into hyperrectangles. We experiment on the performance of these algorithms in order to find the best possible algorithm, or combination thereof.
- **Contribution 2:** Taking advantage of the structure of the models used in LSAT, we propose an approach to perform parametric critical-path analysis of parameterised LSAT models.

1.5 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 presents related work in the literature. Chapter 3 illustrates the modeling concepts of LSAT using an example model. It further introduces the mathematical notations and problem definitions that are used throughout the thesis. Chapters 4 and 5 illustrate the polytope-based and hyperrectangle-based characterization algorithms, respectively, by solving an example characterization problem. These chapters further provide pseudocodes and detailed descriptions of the algorithms. Chapter 6 provides the results of the experiments on the performance of the algorithms and the corresponding conclusions. Chapter 7 summarizes the critical-path analysis of the non-parameterized models that is performed by LSAT. It also provides the description of the proposed approach for the parametric critical-path analysis of the parameterised LSAT models. Finally, Chapter 8 concludes the thesis and discusses possible future work.

Chapter 2

Related Work

In this thesis, our focus is on the parameter-space characterization for the critical path(s) of a parameterized LSAT model. [46] targets the parametric critical-path analysis for event networks with minimal and maximal timelags. The structure of these event networks is highly similar to the structure of the activity models of LSAT, essentially being directed acyclic graphs. [46] proposes an efficient algorithm for characterizing the duration (length) of critical path(s) in event models with timelags that are affine functions of parameters. The result is a set of *convex* regions partitioning the parameter space and a corresponding expression for the duration of the critical path(s). The approach allows an arbitrary number of parameters, with rational numbers as values and weights. The convexity of critical-path regions allows to characterize a region by only analysing the corner points. Most of the methods proposed for the parametric analysis of critical paths with multiple parameters ([46], [24], [27], [19]) are based on the convexity of performance regions. Parameterised LSAT models greatly depend on nonlinear behavior such as motion profiles [35]. Because of this nonlinear behavior, such convexity of performance regions is generally not satisfied. Therefore, direct application of methods based on convexity of performance regions is not possible.

From [46], we observe that the parametric critical-path problem can be transformed to a system of inequality constraints. A method for solving such systems with no limitation to convexity is proposed in [31]. The focus of this work is on the approximation of regions defined by inequality constraints. Since it is not possible to efficiently find the exact boundaries of the regions in the parameter space for general nonlinear inequality constraints, the work proposes an algorithm that estimates the region with a bounded error using interval analysis (which is a mathematical technique that represents the value of a function as a range of possibilities, including the actual value, rather than a single value). The algorithm (SIVIA) is based on partitioning of the parameter space into sets of hyperrectangles and enclosing the region between internal and external unions of these hyperrectangles. Using inclusion functions, each hyperrectangle is classified as feasible, infeasible or indeterminate. Feasible hyperrectangles are guaranteed to only include parameter values that satisfy the inequality constraints while infeasible hyperrectangles are guaranteed to only include parameter values that do not satisfy the inequality constraints. No such guarantee can be given for the indeterminate hyperrectangles. The algorithm continues until each hyperrectangle is classified as feasible, infeasible, or indeterminate, where the indeterminate ones have a size smaller than a given threshold. This threshold determines the error that bounds the accuracy of the estimation. Because of

no limitation to convexity and the general applicability of interval arithmetic to functions with different characteristics, one version of the hyperrectangle-based algorithms, presented in Chapter 5, in this thesis is an adaptation of SIVIA.

In the execution of SIVIA, an increased number of partitions results in an increased number of hyperrectangles that needs to be analysed. This can significantly increase the computational complexity. To this end, algorithms called contractors are proposed. Contractors are used to decrease the size of a hyperrectangle by identifying the parts of it that are infeasible. This makes it possible to obtain a smaller hyperrectangle without partitioning it, leading to analysis of one smaller hyperrectangle rather than two smaller hyperrectangles. In the book [30], several contractors are introduced. Some examples are based on Gauss elimination, forward-backward propagation, and linear programming. These contractors are further generalized to deal with a much larger class of problems and made to collaborate in order to increase their efficiency. It is stated that no contractor is universally better than the others and their efficiency depends on the problem at hand such as the properties of equalities/inequalities and size of boxes. Some other works that discuss contractors are [29], [25], [48], [5], [39], [7], [28], [4], [3], [17]. The use of contractors may also be beneficial for our characterization algorithms. However, we did not yet explore this. This is an interesting direction for future work.

Another field of research that works on nonlinear systems of inequality constraints is parametric programming. [8] introduces multi-parametric programming as "an optimisation based methodology which systematically characterises the effect of uncertain parameters on the optimal solution of mathematical programming problems". Our interest is multi-parametric nonlinear programming (mp-NLP). [8] classifies the work in the literature by convex problems and nonconvex problems. For convex mp-NLP, [33] proposes an approximate algorithm by partitioning the parameter space into a set of hyperrectangles. Using linear interpolation, a fixed-point NLP problem is solved for each corner of the hyperrectangles. The partitioning is done with a prespecified tolerance in order to increase the accuracy of the solutions. [2] proposes a similar algorithm but, instead of hyperrectangles, it partitions the parameter space using simplices. For parameter spaces defined by nonlinear inequalities, [10] uses local mp-QPs/LPs (Q stands for quadratic and L for linear) approximations of the mp-NLP. For nonconvex mp-NLP, [11] proposes different parametric convex over-approximations with a branch-and-bound algorithm. [12] focuses on mp-NLPs with polynomial nonlinear terms and uses Cylindrical Algebraic Decomposition [13] and Homotopy Continuation methods [14] to find analytical solutions. [22] transforms mp-NLP into a system of simultaneous nonlinear equations and a follow-up work [23] approximates the solutions of these nonlinear systems using parameterised triangulations. [37] introduces subgradient-approaches while [9] presents an exact solution for the case of exponential nonlinear terms.

In general, characterization problems cannot be classified as parametric programming problems since parametric programming is focused on optimisation. However, in parametric programming, the uncertainty in parameters is specified using systems of equalities/inequalities and optimisation is done under these constraints. Such systems form the basis of our characterization problem as it is presented in [46]. In our work, we do not directly use any methods from parametric programming. However, we were inspired by the fact that parametric programming methods for mp-NLPs are highly dependent on affine and/or quadratic approx-

imations. In the polytope-based characterization algorithm, presented in Chapter 4, we use such affine approximations to be able to use methods in the literature, designed for parametric critical-path problems with convex performance regions, for our parametric critical-path problem with nonconvex performance regions. Specifically, by using affine approximations of nonlinear expressions in the inequality constraints, we use the approach of [46] which does characterization of a region by only analysing the corner points.

With respect to approximation, we are specifically interested in approximation techniques for nonlinear functions that can provide a bound on the accuracy of the approximation, because these can enable solutions with guaranteed accuracy for our characterization problem. [40] introduces an algorithm that finds piecewise-affine under- and over-approximations of functions with one or two variables over triangular regions with arbitrary precision. For certain types of functions with more than two variables, the work proposes transformations that enable the use of one- and two-dimensional approximation techniques. [1] finds arbitrarily precise piecewise-affine under- and over-approximations of Lipschitz-continuous nonlinear functions by partitioning the parameter space with hyperrectangles and finding a pair of affine under- and over-approximations for each hyperrectangle. If the accuracy of the approximations is below a given threshold then the hyperrectangle is split into smaller hyperrectangles and approximation is done again for these smaller hyperrectangles. The algorithm continues until approximations with desired accuracy are found for all the hyperrectangles. The approximations are done by using the Lipschitz constant of the function. [32] finds such approximations by partitioning the parameter space with polytopes. It uses mesh-based approximation techniques that can be used for functions with different differentiability properties rather than only Lipschitz-continuous functions. However, this creates complications as finding a suitable mesh for a polytope is not as straightforward as finding a mesh for a hyperrectangle. It tackles this issue by finding a bigger polytope that encapsulates the polytope in consideration and making the approximation according to the mesh of this bigger polytope. In the polytope-based characterization algorithm, we use the approach of [32] to find affine approximations of the nonlinear duration functions of potentially critical paths.

Chapter 3

Preliminaries

3.1 Modeling Concepts of LSAT

In this section, we present a simple LSAT model that is also used in Chapters 4 and 5 as a motivating example. We demonstrate only a small part of the capabilities of LSAT as our main goal is to show the structure of activity diagrams. A detailed explanation about the theory behind LSAT can be found in [43] and [44] and a guide for the tool is available on the official website of LSAT, https://www.eclipse.org/lsat/.

An *activity* is a piece of functionally determinate behavior that consists of *actions* executed on *peripherals* of *resources*, in some specified order. Action-level concurrency is allowed, as long as the functional behavior is determinate. A *machine* specification is used to specify the resources with their peripherals. Two types of peripherals are *unmovable* and *movable* peripherals. Unmovable peripherals only declare actions. Movable peripherals (that can also declare actions) are peripherals that change their physical location. For the specification of the physical location change, *SetPoints* define a physical coordinate system. *Axes* relate to the symbolic coordinate system on which the physical locations are applied.

Figure 3.1 shows a part of the machine specification for our LSAT example model that specifies the peripheral types. **Clamp** is an example of an unmovable peripheral as it only declares actions **clamp** and **unclamp** while **Motor** is a movable peripheral with defined **SetPoints** and **Axes**. Any number of **SetPoints** can be specified on the corresponding **Axes**.

A *resource* defines its peripherals with specification of their symbolic positions and paths if the peripheral is of type movable. *SymbolicPositions* declare the symbolic positions for a movable peripheral, *Profiles* declare the speed profiles to use for the paths and *Paths* declare which moves are allowed in the system with the corresponding speed profile.

Figure 3.2 shows a fragment of the machine file of our example that specifies resources using instantiations of peripherals in Figure 3.1. **Robot1** includes a peripheral **P1** of type **Motor** and **SymbolicPositions** declares the symbolic positions as **Right** and **Left**. **Profiles** declare the speed profile **normal** and **Paths** declare two paths between **Right** and **Left**. **Robot2** and **Robot3** only include one peripheral **P2** and **P3**, respectively, of type **Clamp**. It should be noted that this model does not represent an actual machine as, in general, motors

are combined with some other peripheral(s) that they move and peripherals such as clamps are combined with a motor as they cannot move on their own. Such details are ignored in the example to keep the model as simple as possible.

```
Machine Simple

PeripheralType Motor {
    Actions{
        apply_brake
    }
    SetPoints{
        X
    }
    Axes{
        X moves X
    }
}

PeripheralType Clamp{
    Actions{
        clamp
        unclamp
    }
}

```

Figure 3.1: Peripherals

```
Resource Robot1{
    P1: Motor{
        SymbolicPositions{
            Right
            Left
        }
        Profiles (normal)
        Paths{
            Right --> Left profile normal
            Left --> Right profile normal
        }
    }
}

Resource Robot2{
    P2: Clamp
}

Resource Robot3{
    P3: Clamp
}

```

Figure 3.2: Resources

To specify action durations, we need timings. A timing can be a *fixed value*, a *normal* distribution, a *triangular* distribution or a *Pert* distribution. A *motion profile* needs to be assigned to each speed profile. By default, LSAT uses *third-order point-to-point* motion profiles, which need specification of velocity, acceleration, jerk and (optionally) settling time. A user can also specify a custom motion profile. *Positions* specify the physical absolute locations.

Figure 3.3 shows the specifications of the timings for the peripherals and their actions. Action **unclamp** of **Robot2.P2** is specified using a **normal distribution** while the rest of the actions are specified using **fixed values**. For the movable peripheral **Robot1.P1**, we define a motion profile for its speed profile **normal** and physical relative locations to **Right** and **Left**. We use the default **third-order point-to-point motion profile** provided by LSAT (where **V** is velocity, **A** is acceleration, **J** is jerk and **S** is settling time).

Activity specifies activities by defining individual actions and dependencies among these actions. *Prerequisites* specifies the initial location for all movable peripherals for this activity. *Actions* can specify four different types. Resource claiming is done with *claim* actions while resource releasing is done with *release* actions. The timings of these types of actions are equal to zero. Other types of actions are *peripheral* action and *move* action. *Action flow* specifies dependencies between actions by using arrows.

Figure 3.4 shows the specification of a simple activity. The initial location of **Robot1.P1** is defined to be **Left**. Resources need to be claimed before their actions can be executed (the cl*i* actions in the figure) and released (the rl*i* actions) in order to complete the activity. Action **a1** is a **move** action of **Robot1.P1** from **Left** to **Right** using the defined speed profile **normal**. Actions **a2** and **a3** are peripheral actions **clamp**. The arrows in **action flow** specify the precedence relations between actions such as action **a1** cannot start until **cl1** finishes.

The graphical representation of the activity is shown in Figure 3.5.

```
Robot1.P1{
    Timings{
        apply_brake = 100e-3
    }
    Axis X{
        Profiles{
            normal(V = 0.5, A = 0.5, J = 0.5, S = 0.1)
        }
        Positions{
            Left = 0
            Right = 200
        }
    }
}

Robot2.P2{
    Timings{
        clamp = 0.9
        unclamp = Normal(mean = 1.1, sd = 0.1)
    }
}

Robot3.P3{
    Timings{
        clamp = 0.9
        unclamp = 0.8
    }
}
```

Figure 3.3: Settings

```
activity ExampleActivity{
    prerequisites{
        Robot1.P1 at Left
    }
    actions{
        cl1: claim Robot1
        rl1: release Robot1
        cl2: claim Robot2
        rl2: release Robot2
        cl3: claim Robot3
        rl3: release Robot3

        a1: move Robot1.P1 to Right with speed profile normal
        a2: Robot2.P2.clamp
        a3: Robot3.P3.clamp
    }
    action flow{
        cl1 -> a1 -> rl1
        cl2 -> a2 -> rl2
        cl3 -> a3 -> rl3
    }
}
```

Figure 3.4: Activity

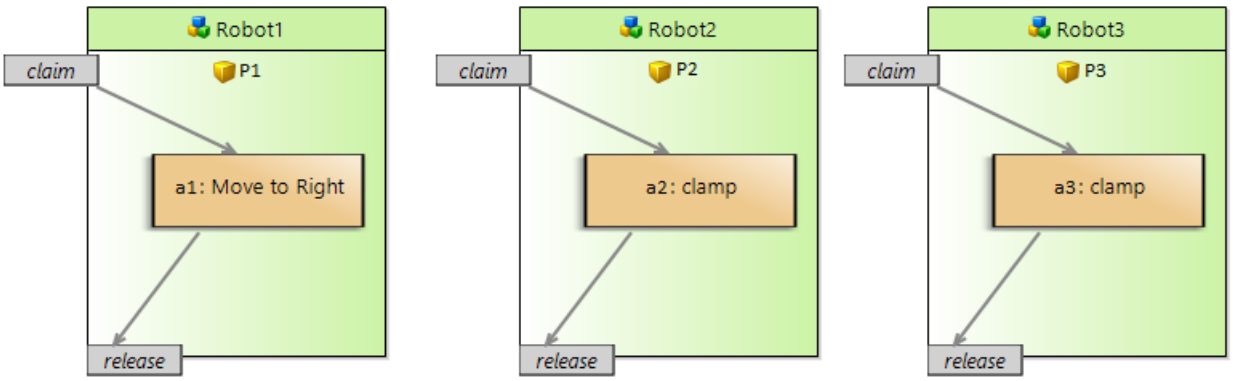

Figure 3.5: Graphical Representation of an Activity

3.2 Notations

This section introduces the definitions that are used in the following chapters.

- A *directed graph* $DG(N, \rightarrow)$ consists of a set N of *nodes* and a set $\rightarrow \subseteq N$ of *dependencies*. A dependency $(n_1, n_2) \in \rightarrow$ is written as $n_1 \rightarrow n_2$ and defines node n_1 as a predecessor of node n_2. A node that has no predecessor is called a *source node*. A node that is not a predecessor of any node is called a *sink node*.
- A *path* p of length $k \in \mathbb{N}$ in a directed graph $DG(N, \rightarrow)$ is a sequence of nodes $p = (n_0, n_1, \ldots, n_k)$ such that $n_i \in N$ for $i = 0, 1, ..., k$ and $n_j \rightarrow n_{j+1}$ for $j = 0, 1, ..., k-1$; nodes n_i are said to be elements of p, denoted $n_i \in p$.
- A *cycle* in a directed graph $DG(N, \rightarrow)$ is a path with length $k > 0$ and the additional constraint that the first and last nodes are equal.
- A *directed acyclic graph* $DAG(N, \rightarrow)$ is a directed graph $DG(N, \rightarrow)$ with no cycles.
- A *d-polyhedron* is the intersection of finitely many *half-spaces* in $\mathbb{R}^d$.
- A *d-polytope* is a bounded d-polyhedron.
- A set of sets Par is a *partition* of a set X if and only if all of the following conditions hold:
  - $\emptyset \notin Par$
  - $\bigcup_{A \in Par} A = X$ where each $A \in Par$ is called a *block*
  - $A_i \cap A_j = \emptyset$ for all $A_i, A_j \in Par$ with $i \neq j$
- An *affine over-approximation* $\overline{f}$ of a function f over domain P is an affine function such that $\overline{f}(x) \geq f(x)$ for all $x \in P$.
- An *affine under-approximation* $\underline{f}$ of a function f over domain P is an affine function such that $\underline{f}(x) \leq f(x)$ for all $x \in P$.

3.3 Parametric Critical-Path Problem

We focus on the parametric critical-path analysis of a directed acyclic graph. The adopted notation is given as follows:

- Let $\mathbb{D} \subset \mathbb{R}^n$ be a bounded *parameter space* where n is the number of parameters.
- A node $n \in N$ of a directed acyclic graph $DAG(N, \rightarrow)$ is associated with a continuous *duration function* $d_n : \mathbb{D} \rightarrow \mathbb{R}_{\geq 0}$.
- A *maximal path* in a directed acyclic graph $DAG(N, \rightarrow)$ is a path in which the first node is a source node and the last node is a sink node.
- Let $\mathcal{P}_A$ be the *set of all maximal paths* in a given directed acyclic graph $A = DAG(N, \rightarrow)$.
- A path p is associated with a *duration function* $f_p : \mathbb{D} \rightarrow \mathbb{R}_{\geq 0}$ that is the summation of the duration functions of all the nodes included in $p : f_p = \sum_{n \in p} d_n$.
- A maximal path $p_c \in \mathcal{P}_A$ of a directed acyclic graph A is a *critical path* at point $x \in \mathbb{D}$ if and only if $f_{p_c}(x) = \max_{p \in \mathcal{P}_A} f_p(x)$.

We define the parametric critical-path problem as follows:

Given a directed acyclic graph A and a bounded parameter space $\mathbb{D}$, find the mapping $PCP_A : \mathbb{D} \rightarrow \wp(\mathcal{P}_A) \setminus \{\emptyset\}$, where $\wp(\mathcal{P}_A)$ is the power set of $\mathcal{P}_A$, that associates each parameter point $x \in \mathbb{D}$ to its critical path(s).

3.4 Parameterised LSAT Model

The activity model given in Figure 3.5 is equal to the directed acyclic graph $A = DAG(N_A, \rightarrow_A)$ given in Figure 3.6.

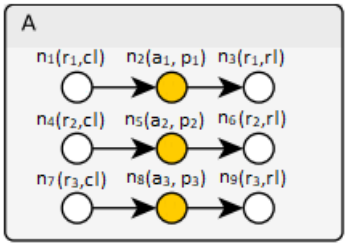

Figure 3.6: DAG of the activity model

Each *action node* $n \in N_A$ is mapped on to an action with the corresponding peripheral; for example, n_2 is mapped on to (a_1, p_1). The *claim nodes* n_1, n_4 and n_7 are mapped on to claim actions while the *release nodes* n_3, n_6 and n_9 are mapped on to release actions. Claim and release actions have a corresponding resource rather than a peripheral. It should be noted that resources Robot1, Robot2 and Robot3 are represented with r_1, r_2 and r_3, respectively.

As shown in Figure 3.3, the durations of actions in the model are specified concretely, by fixed values, motion profiles, or distributions. Now, we parameterise this model by assigning duration functions to actions. No duration function is assigned to claim and release actions considering that LSAT prescribes that their durations are always fixed to 0. The duration functions assigned to the other actions are the following:

$$d_{a_1}(x,y) = x^3 + y^3 - 20x - 20y + 150 \tag{3.1}$$

$$d_{a_2}(x,y) = -x^3 + y^2 + 20x + y + 110 \tag{3.2}$$

$$d_{a_3}(x,y) = x^2 + 130 \tag{3.3}$$

Note that these duration functions are chosen to illustrate the characterization algorithms in later chapters. They do no represent any meaningful system.

By defining the duration functions d_n of nodes $n \in N_A$ to be equal to the duration functions of the corresponding actions, we get the following duration functions:

$$d_{n_1}, d_{n_3}, d_{n_4}, d_{n_6}, d_{n_7}, d_{n_9} = 0 \tag{3.4}$$

$$d_{n_2}(x,y) = d_{a_1}(x,y) = x^3 + y^3 - 20x - 20y + 150 \tag{3.5}$$

$$d_{n_5}(x,y) = d_{a_2}(x,y) = -x^3 + y^2 + 20x + y + 110 \tag{3.6}$$

$$d_{n_8}(x,y) = d_{a_3}(x,y) = x^2 + 130 \tag{3.7}$$

The set of all maximal paths in A is $\mathcal{P}_A = \{p_1, p_2, p_3\}$ where $p_1 = (n_1, n_2, n_3), p_2 = (n_4, n_5, n_6)$ and $p_3 = (n_7, n_8, n_9)$. Then, the duration functions of these maximal paths are $f_{p_1} = d_{n_1} + d_{n_2} + d_{n_3} = d_{n_2}, f_{p_2} = d_{n_4} + d_{n_5} + d_{n_6} = d_{n_5}$ and $f_{p_3} = d_{n_7} + d_{n_8} + d_{n_9} = d_{n_8}$. Let the parameter space of interest for this activity be $\mathbb{D}_A = \mathbb{D}_x \times \mathbb{D}_y$ where $\mathbb{D}_x = [3,6]$ and $\mathbb{D}_y = [3,6]$. Then, the parametric critical-path problem for A is to find the mapping $PCP_A : \mathbb{D}_A \to \wp(\mathcal{P}_A) \setminus \{\emptyset\}$.

3.5 Characterization Problem

This section defines a generic characterization problem. Assuming that the duration functions of maximal paths in a DAG are known, the parametric critical-path problem can be solved by solving the corresponding characterization problem.

- Let $\mathbb{P} \subset \mathbb{R}^n$ be a bounded parameter space where n is the number of parameters.
- Let $\mathcal{F}$ be a set of performance functions of the form $f : \mathbb{P} \to \mathbb{R}$.
- A performance function $f_d \in \mathcal{F}$ is a *dominant function* at point $x \in \mathbb{P}$ if and only if $f_d(x) = \max_{f \in \mathcal{F}} f(x)$.

We define the characterization problem as follows:

Given a parameter space $\mathbb{P}$ and a function set $\mathcal{F}$, find the mapping $C : \mathbb{P} \to \wp(\mathcal{F}) \setminus \{\emptyset\}$ that associates each parameter point $x \in \mathbb{P}$ to the function(s) $f \in \mathcal{F}$ that are dominant at point x.

The solution of the parametric critical-path problem $PCP_A : \mathbb{D} \to \wp(\mathcal{P}_A) \setminus \{\emptyset\}$ is equivalent to the solution of the characterization problem $C : \mathbb{P} \to \wp(\mathcal{F}) \setminus \{\emptyset\}$ where $\mathbb{P} = \mathbb{D}$ and $f_p \in \mathcal{F}$ for all $p \in \mathcal{P}_A$. Therefore, if we obtain the duration functions of all the maximal paths in A, then we can solve the corresponding characterization problem to solve the parametric critical-path problem.

The characterization problem for the example parameterised activity model is to find the mapping $C : \mathbb{P} \to \wp(\mathcal{F}) \setminus \{\emptyset\}$ with $\mathbb{P} = \mathbb{P}_x \times \mathbb{P}_y$ and $f_1, f_2, f_3 \in \mathcal{F}$ where $\mathbb{P}_x = [3,6]$, $\mathbb{P}_y = [3,6]$ and $f_1(x,y) = x^3 + y^3 - 20x - 20y + 150, f_2(x,y) = -x^3 + y^2 + 20x + y + 110$ and $f_3(x,y) = x^2 + 130$.

3.6 Third-Order Point-To-Point Motion Profiles

This section presents the third-order point-to-point motion profile that is used by default in LSAT. Given distance dis and maximum levels of velocity $vMax$, acceleration $aMax$ and jerk $jMax$, the duration of the third order point-to-point motion profile $f(dis, vMax, aMax, jMax)$ is given as follows:

1: //Calculation of $f(dis, vMax, aMax, jMax)$ is done according to the following constraints:

2: **if** $aMax - \sqrt{vMax \cdot jMax} < 0$ **then**
3: **if** $dis - \frac{vMax^2}{aMax} - \frac{vMax \cdot aMax}{jMax} > 0$ **then**
4: $f(dis, vMax, aMax, jMax) = \frac{aMax}{jMax} + \frac{vMax}{aMax} + \frac{dis}{vMax}$
5: **else if** $dis - \frac{2aMax^3}{jMax^2} > 0$ **then**
6: $f(dis, vMax, aMax, jMax) = \frac{aMax}{jMax} + \sqrt{\frac{aMax^2}{jMax^2} + \frac{4dis}{aMax}}$
7: **else**
8: $f(dis, vMax, aMax, jMax) = 4\sqrt[3]{\frac{dis}{2jMax}}$
9: **end if**
10: **else**
11: **if** $dis - 2\sqrt{\frac{vMax^3}{jMax}} < 0$ **then**
12: $f(dis, vMax, aMax, jMax) = 4\sqrt[3]{\frac{dis}{2jMax}}$
13: **else**
14: $f(dis, vMax, aMax, jMax) = 2\sqrt{\frac{vMax}{jMax}} + \frac{dis}{vMax}$
15: **end if**
16: **end if**

Considering that the values of $dis, vMax, aMax$ and $jMax$ are positive, the left-hand side expressions of the constraints (Lines 2, 3, 5, 11) are monotone; that is, the expressions' first-order partial derivatives with respect to all variables $(dis, vMax, aMax, jMax)$ are always positive or always negative. The right-hand side expressions in Lines 4, 6, 8, 12 and 14 are also monotone for the domain of variables that satisfy the corresponding constraint. For example, the expression $\frac{aMax}{jMax} + \frac{vMax}{aMax} + \frac{dis}{vMax}$ is monotone for variables that satisfy $dis - \frac{vMax^2}{aMax} - \frac{vMax \cdot aMax}{jMax} > 0$. Furthermore, these right-hand side expressions' first-order partial derivatives with respect to all variables are monotone too.

Chapter 4

Polytope-Based Characterization

In this chapter and the following one, we present different algorithms that solve the characterization problem introduced in Section 3.5. Manufacturing systems depend on physical subsystems that transport the material from one processing station to another. The modeling of such physical movement is an important part of FMS design. By default, LSAT models use third-order point-to-point motion profiles (given in Section 3.6). The presented polytope- and hyperrectangle-based algorithms are specifically targeted towards characterization of models using the third-order point-to-point motion profiles as action duration functions. Therefore, the (monotonicity) properties required by the algorithms are motivated by the properties of the third-order point-to-point motion profiles. Considering that LSAT models do not only use the third-order point-to-point motion profiles, for generic applicability, we also implement a hyperrectangle-based algorithm that uses interval arithmetic. The algorithms are illustrated by solving the earlier introduced example characterization problem given in Section 3.5. To facilitate understanding, in this example problem, we use the simple performance functions introduced earlier rather than the complex third-order point-to-point motion profiles.

In this chapter, we demonstrate the polytope-based characterization algorithm (polytope algorithm, for short). In the first section, we solve the earlier introduced example characterization problem using the polytope algorithm. In the second section, we provide the pseudocode for the polytope algorithm and provide further details.

4.1 Polytope Algorithm Example

The polytope algorithm is illustrated by solving the following characterization problem:
Find the mapping $C : \mathbb{P}_0 \to \wp(\mathcal{F}) \setminus \{\emptyset\}$ with $\mathcal{F} = \{f_1, f_2, f_3\}$ and $\mathbb{P}_0 = \{p \in \mathbb{R}^2 \mid Gp \geq g\}$ where

$$f_1(x, y) = x^3 + y^3 - 20x - 20y + 150$$
$$f_2(x, y) = -x^3 + y^2 + 20x + y + 110$$
$$f_3(x, y) = x^2 + 130$$

$$G = \begin{bmatrix} \frac{-2}{3} & -1 \\ -2 & 1 \\ \frac{3}{4} & 1 \\ 1 & -1 \end{bmatrix} \quad g = \begin{bmatrix} -9 \\ -7 \\ \frac{27}{4} \\ \frac{-3}{2} \end{bmatrix}$$

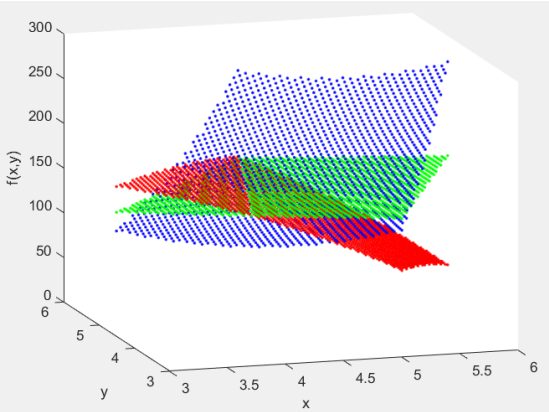

Figure 4.1: Sampled performance functions blue:$f_1(x, y)$, red:$f_2(x, y)$, green:$f_3(x, y)$

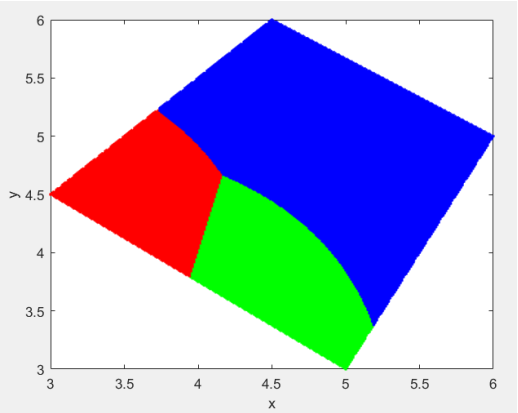

Figure 4.2: Sampled exact solution of the characterization problem blue:f_1, red:f_2, green:f_3

This characterization problem is the one presented in Section 3.5 with a change in the parameter space $\mathbb{P}$. This change is applied to show that the polytope algorithm allows arbitrary polytopes as parameter space rather than only hyperrectangles, which are a specific type of polytope.

Example 4.1.1 Figure 4.1 shows sampled versions of f_1, f_2 and f_3 for the parameter space $\mathbb{P}_0$. A sampled version of the exact solution of this characterization problem is given in Figure 4.2. The blue region is the part of the parameter space with f_1 being the only dominant performance function. Only f_2 is dominant in the red region and only f_3 is dominant in the green region. The only exceptions are the borders between regions. On the borders where two different regions meet, both performance functions are dominant. At the point where all three regions meet, all of the performance functions are dominant.

To solve the characterization problem, the polytope algorithm uses affine over- and under-approximations of the performance functions. We calculate these affine approximations by using the approach of Jin et al. [32]. We assume that the partial derivatives of performance functions with respect to all of their parameters are monotone. This is needed to efficiently compute the approximations. Without the monotonicity assumption, we would require optimization techniques that are computationally expensive. The assumption of monotonicity is further motivated by the monotonicity properties of the third-order point-to-point motion profile.

Example 4.1.2 In the presented characterization problem, the partial derivatives of performance functions f_1, f_2 and f_3 with respect to parameters x and y are monotone on polytope $\mathbb{P}_0$.

The first step of the polytope algorithm is to calculate affine over- and under-approximations of the performance functions over the entire parameter space.

Example 4.1.3 For the given characterization problem, the algorithm first calculates affine over- and under- approximations of f_1, f_2 and f_3 over $\mathbb{P}_0$. We demonstrate this calculation by

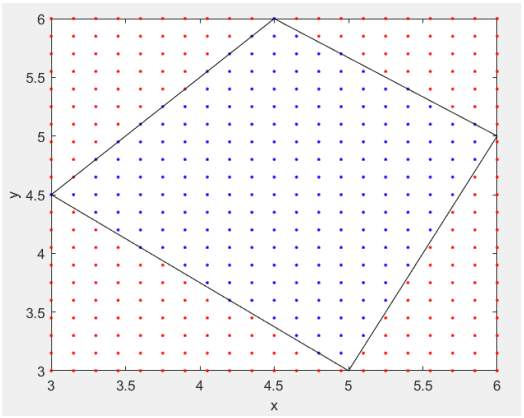

Figure 4.3: Uniform mesh for $\mathbb{EH}_{0,1}$ and $IGP_{0,1}$ (blue)

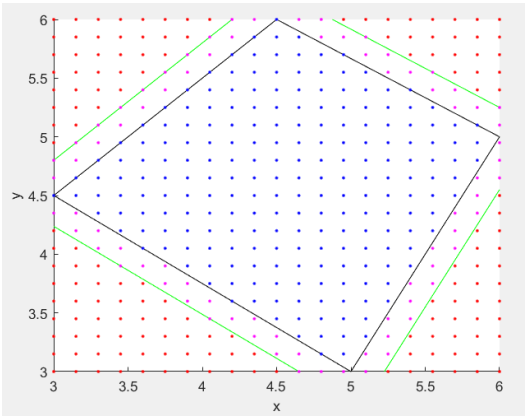

Figure 4.4: Extended boundaries of the parameter space $\mathbb{P}_0$ (green), $OGP_{0,1}$ (magenta) and $GP_{0,1}$ (magenta and blue)

showing the steps for calculating the approximations of performance function f_1. Performance function f_1 has both parameters (x and y) of $\mathbb{P}_0$. Therefore, we use the polytope parameter space $\mathbb{P}_{0,1} = \mathbb{P}_0$ to calculate approximations of f_1 over $\mathbb{P}_0$. We first find the smallest hyperrectangle that encapsulates the polytope parameter space $\mathbb{P}_{0,1}$. This smallest encapsulating hyperrectangle $\mathbb{EH}_{0,1} = \{(x, y) \mid x \in [3, 6],\ y \in [3, 6]\}$ can be seen in Figure 4.2 as the accumulation of the white and colored regions. We need a mesh of grid points to find the affine approximations. By using this encapsulating hyperrectangle $\mathbb{EH}_{0,1}$, we create a uniform mesh, because directly creating a non-uniform mesh for a polytope could be computationally expensive. Figure 4.3 shows a uniform mesh for $\mathbb{EH}_{0,1}$. The set $IGP_{0,1}$ includes the blue colored grid points that are inside or on the border of $\mathbb{P}_{0,1}$. The number of grid points along each dimension (x and y) or resolution is equal to 21 for this mesh. In general, a different resolution can be chosen for different dimensions. If we use all the grid points to find the affine approximations, then we obtain approximations of f_1 over $\mathbb{EH}_{0,1}$. Considering that we want to find affine approximations over $\mathbb{P}_{0,1}$, these approximations over the entire $\mathbb{EH}_{0,1}$ are suboptimal over $\mathbb{P}_{0,1}$. Therefore, rather than using all the grid points, we eliminate some of the points that are not needed to obtain approximations over $\mathbb{P}_{0,1}$. If we only use the blue grid points that are inside or on the border of $\mathbb{P}_{0,1}$, then the resulting approximations may not be applicable over the entire $\mathbb{P}_{0,1}$. Therefore, we create a larger polytope that encapsulates $\mathbb{P}_{0,1}$ by "extending its boundaries outwards by the distance between adjacent grid points in each dimension such that the enlarged polytope $\tilde{\mathbb{P}}_{0,1}$ is guaranteed to include the immediate neighbor grid points to the boundaries" [32]. Figure 4.4 shows the extended boundaries for our characterization problem using green lines. The set $OGP_{0,1}$ includes the magenta colored grid points that are inside or on the border of $\tilde{\mathbb{P}}_{0,1}$ but not inside or on the border of $\mathbb{P}_{0,1}$. Let set $GP_{0,1}$ be the union of $IGP_{0,1}$ and $OGP_{0,1}$ including both the magenta and blue colored points in Figure 4.4. $GP_{0,1}$ is the set of grid points that we use to calculate over- and under-approximations of f_1.

To solve the characterization problem, the parameter space is partitioned step by step as we continue calculating new affine over- and under-approximations and use these approximations

to characterize as much region as possible. To calculate affine over- and under-approximations over $\mathbb{P}_0$, we solve the following linear program for all $f_j \in \mathcal{F}$ to obtain two affine approximations for each performance function.

$$\begin{aligned} \min_{\theta, \overline{A}_{0,j}, \underline{A}_{0,j}, \overline{h}_{0,j}, \underline{h}_{0,j}} \quad & |\theta| \quad \text{subject to} \\ \overline{A}_{0,j}\, p + \overline{h}_{0,j} &\geq f_j(p), \\ \underline{A}_{0,j}\, p + \underline{h}_{0,j} &\leq f_j(p), \\ (\overline{A}_{0,j} - \underline{A}_{0,j})\, p + \overline{h}_{0,j} - \underline{h}_{0,j} &\leq \theta, \\ \forall p \in GP_{0,j} & \end{aligned} \tag{4.1}$$

The affine approximations obtained from Linear Program 4.1 ($\overline{A}_{0,j}\, p + \overline{h}_{0,j}$ and $\underline{A}_{0,j}\, p + \underline{h}_{0,j}$) for performance function f_j are not over- and under-approximations over the entire continuous $\mathbb{P}_0$. These approximations only hold for the grid points ($GP_{0,j}$), but possibly not for the points in between.

Example 4.1.4 Figure 4.5 shows sampled performance function $f_1 \in \mathcal{F}$ and sampled affine approximations $\overline{A}_{0,1}\, p + \overline{h}_{0,1}$ and $\underline{A}_{0,1}\, p + \underline{h}_{0,1}$ for f_1 where $\overline{A}_{0,1} \approx [44.8\ 43.5]$, $\overline{h}_{0,1} \approx -211$, $\underline{A}_{0,1} \approx [44.8\ 43.5]$, $\underline{h}_{0,1} \approx -245.1$. Figure 4.6 shows the minima of all three functions in Figure 4.5, i.e. a projection along the Z-axis. The blue points show that $\underline{A}_{0,1}\, p + \underline{h}_{0,1} \leq f_1(p)$ is not satisfied for all $p \in \mathbb{P}_0$.

As shown in Equations 4.2 and 4.3, we need quantified approximation errors (σ) to obtain affine over- and under-approximations that are applicable for all $p \in \mathbb{P}_0$. We obtain affine over- and under-approximations over the entire continuous $\mathbb{P}_0$ using the following equations.

$$\overline{A}_{0,j}\, p + \overline{h}_{0,j} + \sigma_{0,j} \geq f_j(p) \tag{4.2}$$

$$\underline{A}_{0,j}\, p + \underline{h}_{0,j} - \sigma_{0,j} \leq f_j(p) \tag{4.3}$$

for $p \in \mathbb{P}_0$ and $f_j \in \mathcal{F}$, where $\overline{A}_{0,j}, \overline{h}_{0,j}, \underline{A}_{0,j}, \underline{h}_{0,j}$ are obtained by solving Linear Program 4.1 for f_j and $\sigma_{0,j}$ is the approximation error of f_j for $\mathbb{P}_0$.

We do not directly calculate $\sigma_{0,j}$. Instead, we use the bound $\delta^*_{0,j} \max_{p \in \mathbb{P}_{0,j}} ||f'_j(p)||_2 \geq \sigma_{0,j}$ from [32] to obtain an approximation of $\sigma_{0,j}$ of each performance function $f_j \in \mathcal{F}$ for $\mathbb{P}_0$. The obtained affine over- and under-approximations are given below.

$$\overline{f}_{0,j} = \overline{A}_{0,j}\, p + \overline{h}_{0,j} + \delta^*_{0,j} \max_{p \in \mathbb{P}_{0,j}} ||f'_j(p)||_2 \geq \overline{A}_{0,j}\, p + \overline{h}_{0,j} + \sigma_{0,j} \geq f_j(p) \tag{4.4}$$

$$\underline{f}_{0,j} = \underline{A}_{0,j}\, p + \underline{h}_{0,j} - \delta^*_{0,j} \max_{p \in \mathbb{P}_{0,j}} ||f'_j(p)||_2 \leq \underline{A}_{0,j}\, p + \underline{h}_{0,j} - \sigma_{0,j} \leq f_j(p) \tag{4.5}$$

where $\delta^*_{0,j} = \delta_{0,j} \sqrt{\frac{n}{2(n+1)}}$ for $\mathbb{P}_{0,j} \subset \mathbb{R}^n$. $\delta_{0,j}$ is the diameter of the mesh elements of the uniform mesh that is used to find the affine approximations for $f_j \in \mathcal{F}$. The diameter of a polytope is the greatest distance between its vertices.

Example 4.1.5 For f_1, we used the uniform mesh given in Figure 4.3 with resolution equal to 21 for both dimensions (x and y). The distance between neighboring grid points for each dimension is equal to $\frac{6-3}{20} = 0.15$ meaning that the diameter of mesh elements is

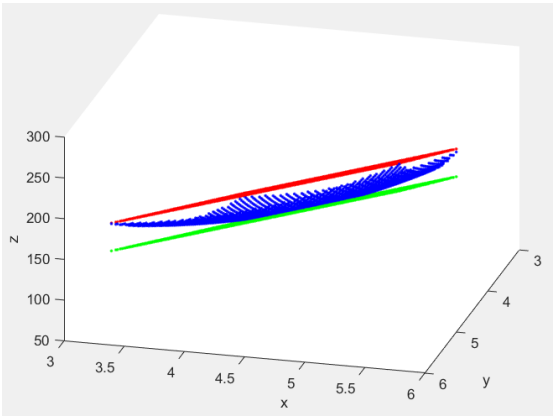

Figure 4.5: Performance function f_1 (blue) and affine approximations $\overline{A}_1 p + \overline{h}_1$ (red) and $\underline{A}_1 p + \underline{h}_1$ (green)

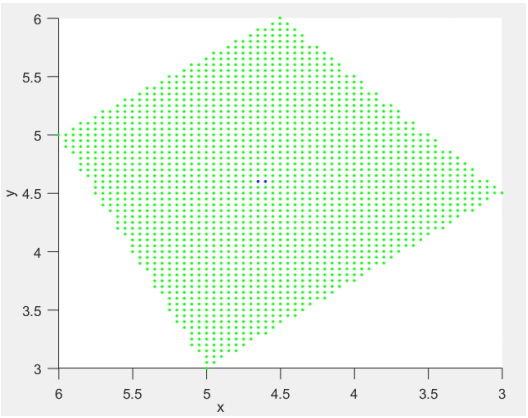

Figure 4.6: Visualization of Figure 4.5 from below (It shows that $\underline{A}_1 p + \underline{h}_1 \leq f_1$ is not satisfied for the entire continuous $\mathbb{P}_0$)

equal to $\delta_{0,1} = \sqrt{0.15^2 + 0.15^2} \approx 0.213$. This results in $\delta^*_{0,1} \approx 0.213\sqrt{\frac{2}{2(2+1)}} \approx 0.123$ considering $\mathbb{P}_{0,1} \subset \mathbb{R}^2$. With the assumption of partial derivatives of performance functions being monotone, we approximate the value of $\max_{p \in \mathbb{P}_{0,j}} ||f'_j(p)||_2$ by using the grid points in $OGP_{0,j}$. Because the extended boundaries are constructed in a way that $\tilde{\mathbb{P}}_{0,j}$ is guaranteed to include the immediate neighbor grid points to the boundaries of $\mathbb{P}_{0,j}$, we know that $\max_{g \in OGP_{0,j}} ||f'_j(g)||_2 \geq \max_{p \in \mathbb{P}_{0,j}} ||f'_j(p)||_2 \geq \sigma_{0,j}$ is satisfied for all $f_j \in \mathcal{F}$. In the example, $\max_{g \in OGP_{0,1}} ||f'_1(g)||_2 \approx 108.05$ and $\sigma_{0,1} \approx 0.123 \cdot 108.05 = 13.3$. Figure 4.7 shows the obtained affine over- and under-approximations $\overline{f}_{0,1}$ and $\underline{f}_{0,1}$ for $f_1 \in \mathcal{F}$. Just as f_1, performance function $f_2 \in \mathcal{F}$ has both parameters (x and y) resulting in $\mathbb{P}_{0,2} = \mathbb{P}_{0,1} = \mathbb{P}_0$. This means that we use the same set of grid points, for the calculation of affine over- and under-approximations of f_2, that we used for f_1 ($GP_{0,2} = GP_{0,1}$). The obtained affine over- and under-approximations of f_2 ($\overline{f}_{0,2}$ and $\underline{f}_{0,2}$) are given in Figure 4.8.

Example 4.1.6 Unlike f_1 and f_2, performance function $f_3 \in \mathcal{F}$ has only parameter x. For f_3, if we use the same set of grid points that we used for f_1 and f_2, then we would use redundant grid points that do not contribute to a better accuracy for $\overline{f}_{0,3}$ and $\underline{f}_{0,3}$. This can simply be seen by the fact that $f_3(3, 3) = f_3(3, 3.15) = f_3(3, 3.3) = ...$ and $||f'_3(3, 4.35)||_2 = ||f'_3(3, 4.5)||_2 = ||f'_3(3, 4.65)||_2 = ...$ Therefore, rather than creating a uniform mesh for the entire two-dimensional $\mathbb{EH}_{0,1}$ as we did for f_1 and f_2, we only need to create a uniform mesh for a one-dimensional subspace. By projecting the two-dimensional vertices of $\mathbb{P}_0$ onto the one-dimensional subspace (x axis) and computing the convex hull of these projected vertices, we find the projection of $\mathbb{P}_0$ onto the x axis. We use this projection, $\mathbb{P}_{0,3}$, to find approximations of f_3 over $\mathbb{P}_0$. The vertices of $\mathbb{P}_0$ are $(4.5, 6), (6, 5), (5, 3), (3, 4.5)$. The projections of these vertices onto the x axis are found by simply removing the y values. Therefore, the projections of these vertices on the x axis are $(4.5), (6), (5), (3)$. The convex hull of these points is $\mathbb{P}_{0,3} = \{x \in \mathbb{R} \,|\, x \geq 3, x \leq 6\}$, which is a one-dimensional polytope. The smallest hyperrectangle that encapsulates this polytope is $\mathbb{EH}_{0,3} = \{x \in \mathbb{R} \,|\, x \geq 3, x \leq 6\}$ considering that a one-dimensional polytope is also a one-dimensional hyperrectangle. The intersection

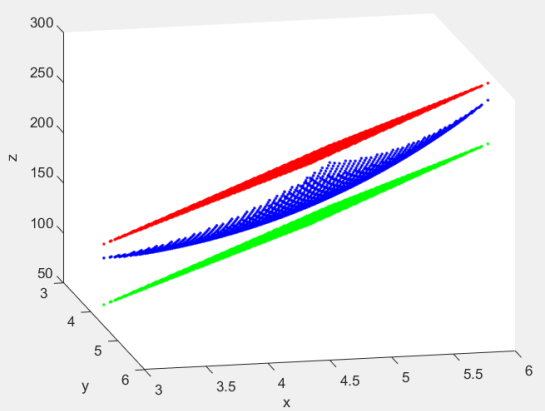

Figure 4.7: Performance function f_1 (blue) and affine over- and under-approximations $\overline{f}_1(p)$ (red) and $\underline{f}_1(p)$ (green)

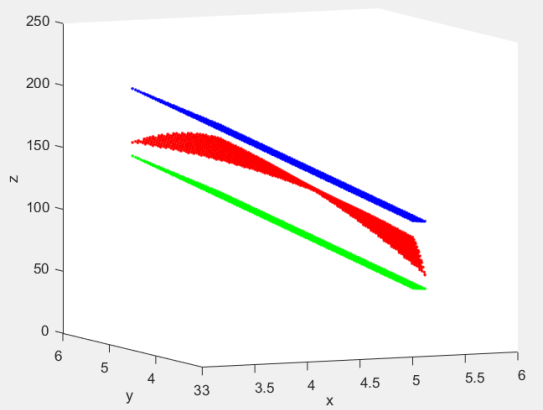

Figure 4.8: Performance function f_2 (red) and affine over- and under-approximations $\overline{f}_2(p)$ (blue) and $\underline{f}_2(p)$ (green)

of the enlarged polytope $\tilde{\mathbb{P}}_{0,3}$, which is obtained by extending the boundaries of polytope $\mathbb{P}_{0,3}$, with the encapsulating hyperrectangle $\mathbb{EH}_{0,3}$ is $\tilde{\mathbb{P}}_{0,3} \cap \mathbb{EH}_{0,3} = \{x \in \mathbb{R} \,|\, x \geq 3, x \leq 6\}$. Therefore, to obtain affine over- and under-approximations of f_3, we use all the grid points in the uniform mesh for the encapsulating hyperrectangle $\mathbb{EH}_{0,3}$, which is given in Figure 4.9. Apart from using a projection of polytope $\mathbb{P}_0$, the other steps of the calculation are the same as for f_1 and f_2. The obtained affine over- and under-approximations $\overline{f}_{0,3}$ and $\underline{f}_{0,3}$ for f_3 are given in Figure 4.10.

Using the obtained affine over- and under-approximations of $f \in \mathcal{F}$, we perform characterization of $\mathbb{P}_0$. Let $\mathcal{F}_0$ be the set of potentially dominant performance functions in $\mathbb{P}_0$. Considering that no characterization is done until now, $\mathcal{F}_0 := \mathcal{F}$. We compare all the different pairs of approximations $(\overline{f}_{0,i}, \underline{f}_{0,j})$ where $f_i, f_j \in \mathcal{F}_0$ for $i \neq j$. Let V_0 be the set of vertices of $\mathbb{P}_0$. In step one for $\mathbb{P}_0$, we check if we can eliminate any performance function that is not dominant at all the points in $\mathbb{P}_0$ (if such a performance function exist). If, for any $f_i, f_j \in \mathcal{F}_0$, $\underline{f}_{0,j}(v) > \overline{f}_{0,i}(v)$ is satisfied for all $v \in V_0$, then it is proven that performance function f_i is not dominant in $\mathbb{P}_0$ resulting in $\mathcal{F}_0 := \mathcal{F}_0 \setminus \{f_i\}$.

Example 4.1.7 We do the characterization of $\mathbb{P}_0$ using the approximations given in Figure 4.7, 4.8 and 4.10. As stated before, because no characterization is done until now, $\mathcal{F}_0 := \mathcal{F} = \{f_1, f_2, f_3\}$. In step one, we cannot do any elimination considering that no function pair $f_i, f_j \in \mathcal{F}_0$ satisfies $\underline{f}_{0,j}(v) > \overline{f}_{0,i}(v)$ for all $v \in V_0$. Therefore, we move on to step two with no change in $\mathcal{F}_0$.

In step two for $\mathbb{P}_0$, using the remaining performance functions in $\mathcal{F}_0$, we check if we can characterize part of $\mathbb{P}_0$. If, for any $f_i, f_j \in \mathcal{F}_0$, $\underline{f}_{0,j}(v) > \overline{f}_{0,i}(v)$ is satisfied for any $v \in V_0$, then it is proven that performance function f_i is not dominant in polytope $\mathbb{P}_1 = \{p \in \mathbb{P}_0 \,|\, \underline{f}_{0,j}(p) > \overline{f}_{0,i}(p)\}$. Therefore, we split $\mathbb{P}_0$ into two smaller polytopes $\mathbb{P}_1 = \{p \in \mathbb{P}_0 \,|\, \underline{f}_{0,j}(p) \geq \overline{f}_{0,i}(p)\}$ and $\mathbb{P}_2 = \{p \in \mathbb{P}_0 \,|\, \underline{f}_{0,j}(p) \leq \overline{f}_{0,i}(p)\}$. It should be noted that, even if it is proven that f_i

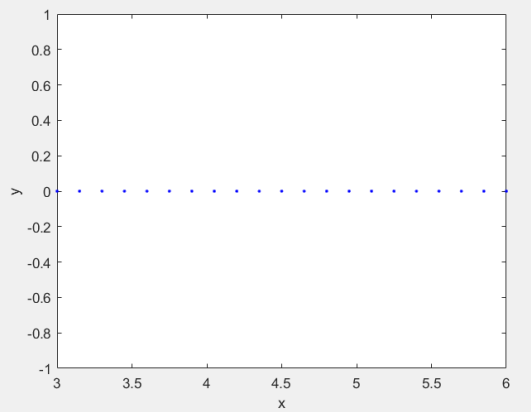

Figure 4.9: Uniform mesh for encapsulating hyperrectangle $x = [3\ 6]$

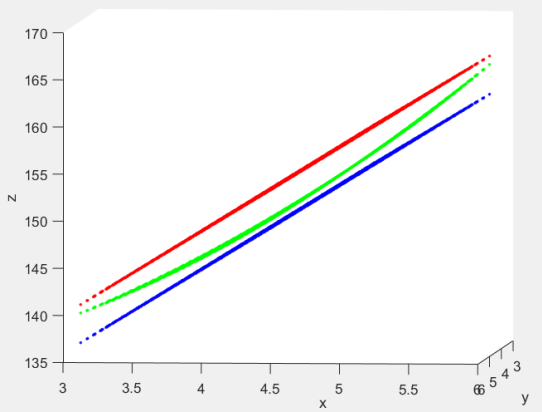

Figure 4.10: Performance function f_3 (green) and affine over- and under-approximations $\overline{f}_3(p)$ (red) and $\underline{f}_3(p)$ (blue)

is not dominant in polytope $\mathbb{P}_1 = \{p \in \mathbb{P}_0 \,|\, \underline{f}_{0,j}(p) > \overline{f}_{0,i}(p)\}$, we use the polytope that includes the borders $\mathbb{P}_1 = \{p \in \mathbb{P}_0 \,|\, \underline{f}_{0,j}(p) \geq \overline{f}_{0,i}(p)\}$. This is required because we need explicit corner points to compare the approximations. As a result, the elimination of performance function f_i is not true for the points that are on the border of $\mathbb{P}_1$. Other than the border, performance function f_i is not dominant in entire $\mathbb{P}_1$ while no elimination is possible for $\mathbb{P}_2$, meaning that $\mathcal{F}_1 := \mathcal{F}_0 \setminus \{f_i\}$ and $\mathcal{F}_2 := \mathcal{F}_0$. Using the approximations we calculated over $\mathbb{P}_0$ $(\overline{f}_{0,i}, \underline{f}_{0,i})$, we repeat steps one and two for $\mathbb{P}_1$ and $\mathbb{P}_2$. In step one for $\mathbb{P}_1$, we check if, for any $f_i, f_j \in \mathcal{F}_1$, $\underline{f}_{0,j}(v) > \overline{f}_{0,i}(v)$ is satisfied for all $v \in V_1$ where V_1 is the set of vertices of $\mathbb{P}_1$. Such $f_i \in \mathcal{F}_1$ is eliminated by $\mathcal{F}_1 := \mathcal{F}_1 \setminus \{f_i\}$. In step two for $\mathbb{P}_1$, we check if, for any $f_i, f_j \in \mathcal{F}_1$, $\underline{f}_{0,j}(v) > \overline{f}_{0,i}(v)$ is satisfied for any $v \in V_1$. If this is satisfied, then $\mathbb{P}_1$ is split into two smaller polytopes $\mathbb{P}_3 = \{p \in \mathbb{P}_1 \,|\, \underline{f}_{0,j}(p) \geq \overline{f}_{0,i}(p)\}$ and $\mathbb{P}_4 = \{p \in \mathbb{P}_1 \,|\, \underline{f}_{0,j}(p) \leq \overline{f}_{0,i}(p)\}$ with $\mathcal{F}_3 := \mathcal{F}_1 \setminus \{f_i\}$ and $\mathcal{F}_4 := \mathcal{F}_1$. It is important to note that we use the approximations $\overline{f}_{0,i}, \underline{f}_{0,i}$, which we calculated over $\mathbb{P}_0$, for the characterization of $\mathbb{P}_1$ even though they are suboptimal for $\mathbb{P}_1$. Our goal is to do all the characterization that is possible by using approximations $\overline{f}_{0,i}, \underline{f}_{0,i}$, which leads to calculating fewer approximations resulting in reduced computation time. That is, we use approximations $\overline{f}_{n,i}, \underline{f}_{n,i}$, calculated over polytope $\mathbb{P}_n$, for the characterization of any polytope $\mathbb{P}_m$ that is a subspace of $\mathbb{P}_n$ until no more characterization can be done using $\overline{f}_{n,i}, \underline{f}_{n,i}$.

Example 4.1.8 In step two for $\mathbb{P}_0$, we find that $\underline{f}_{0,2}(v) > \overline{f}_{0,1}(v)$ for any $v \in V_0$ is satisfied. This is shown in Figure 4.11. Therefore, we split $\mathbb{P}_0$ into two smaller polytopes $\mathbb{P}_1 = \{p \in \mathbb{P}_0 \,|\, \underline{f}_{0,2}(p) \geq \overline{f}_{0,1}(p)\}$ and $\mathbb{P}_2 = \{p \in \mathbb{P}_0 \,|\, \underline{f}_{0,2}(p) \leq \overline{f}_{0,1}(p)\}$ with $\mathcal{F}_1 := \mathcal{F}_0 \setminus \{f_1\} = \{f_2, f_3\}$ and $\mathcal{F}_2 := \mathcal{F}_0$. These polytopes are given in Figure 4.12. For $\mathbb{P}_1$, we repeat step one using $\overline{f}_{0,2}, \underline{f}_{0,2}, \overline{f}_{0,3}$ and $\underline{f}_{0,3}$ considering $\mathcal{F}_1 = \{f_2, f_3\}$. Again, no elimination can be done in step one and we move on to step two with no change in $\mathcal{F}_1$. In step two, we find that $\underline{f}_{0,2}(v) > \overline{f}_{0,3}(v)$ for any $v \in V_1$ is satisfied. Therefore, we split $\mathbb{P}_1$ into two smaller polytopes $\mathbb{P}_3 = \{p \in \mathbb{P}_1 \,|\, \underline{f}_{0,2}(p) \geq \overline{f}_{0,3}(p)\}$ and $\mathbb{P}_4 = \{p \in \mathbb{P}_1 \,|\, \underline{f}_{0,2}(p) \leq \overline{f}_{0,3}(p)\}$ with

$\mathcal{F}_3 := \mathcal{F}_1 \setminus \{f_3\} = \{f_2\}$ and $\mathcal{F}_4 := \mathcal{F}_1$. Because there is only one performance function left in $\mathcal{F}_3$, the characterization of $\mathbb{P}_3$ is complete.

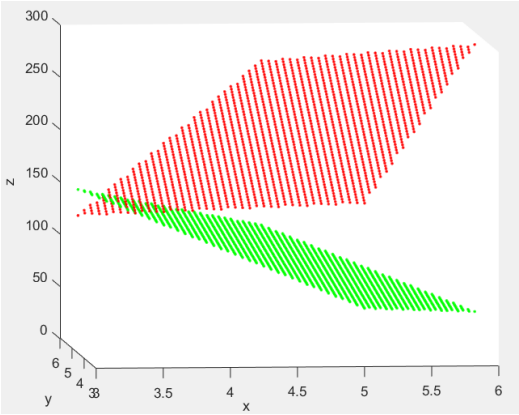

Figure 4.11: Sampled approximations $\overline{f}_{0,2}(p)$ (red) and $\underline{f}_{0,1}(p)$ (green) where $p \in \mathbb{P}_0$

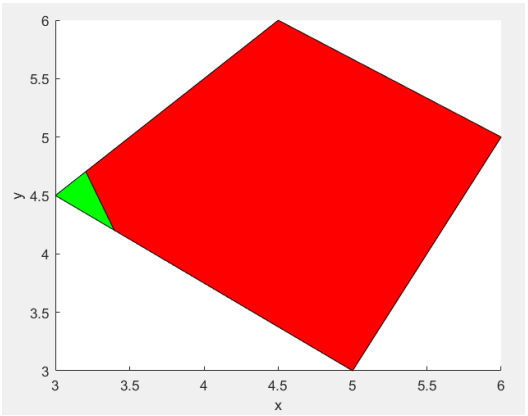

Figure 4.12: Polytopes $\mathbb{P}_1$ (green) and $\mathbb{P}_2$ (red)

If, in step two for $\mathbb{P}_0$, $\underline{f}_{0,j}(v) > \overline{f}_{0,i}(v)$ for any $v \in V_0$ is not satisfied for any pair $f_i, f_j \in \mathcal{F}_0$, then we cannot do any more characterization of $\mathbb{P}_0$ using approximations $\overline{f}_{0,i}, \underline{f}_{0,i}$. Therefore, similar to the previous case, we split $\mathbb{P}_0$ into two smaller polytopes $\mathbb{P}_1$ and $\mathbb{P}_2$. However, different from the previous case, no elimination is done for $\mathbb{P}_1$ and $\mathbb{P}_2$ resulting in $\mathcal{F}_1 := \mathcal{F}_0$ and $\mathcal{F}_2 := \mathcal{F}_0$. Also, no further characterization can be done for $\mathbb{P}_1$ and $\mathbb{P}_2$ using approximations $\overline{f}_{0,i}, \underline{f}_{0,i}$. Therefore, we find new approximations $\overline{f}_{1,i}, \underline{f}_{1,i}$ and $\overline{f}_{2,i}, \underline{f}_{2,i}$ calculated over $\mathbb{P}_1$ and $\mathbb{P}_2$, respectively. Because these approximations are calculated over smaller polytopes, they are more accurate than the approximations we calculated over the bigger polytope (It should be noted that at the worst case, the accuracy of the approximations is the same). Using these more accurate approximations gives us the chance to do further characterization of $\mathbb{P}_1$ and $\mathbb{P}_2$.

Example 4.1.9 For $\mathbb{P}_4$, we repeat steps one and two using $\overline{f}_{0,2}, \underline{f}_{0,2}, \overline{f}_{0,3}$ and $\underline{f}_{0,3}$. No elimination can be done in step one. Also, in step two, $\underline{f}_{0,j}(v) > \overline{f}_{0,i}(v)$ for any $v \in V_4$ is not satisfied for any $f_i, f_j \in \mathcal{F}_4$. This means that we cannot do any further characterization of $\mathbb{P}_4$ using approximations that we found for $\mathbb{P}_0$. Therefore, we continue characterization of $\mathbb{P}_4$ by first calculating new approximations $\overline{f}_{4,2}, \underline{f}_{4,2}, \overline{f}_{4,3}$ and $\underline{f}_{4,3}$ over $\mathbb{P}_4$. These approximations are expected to be more accurate than the approximations that we calculated over $\mathbb{P}_0$. We repeat steps one and two for $\mathbb{P}_4$ using these new approximations $\overline{f}_{4,2}, \underline{f}_{4,2}, \overline{f}_{4,3}$ and $\underline{f}_{4,3}$. The procedure described above is repeated for newly created smaller polytopes until the algorithm terminates. Figure 4.13 shows the total characterization done for $\mathbb{P}_0$ by 150 seconds of execution of the polytope algorithm. The blue, red and green colored polytopes are completely characterized. In the cyan colored polytopes, performance function f_3 is eliminated. In the magenta colored polytopes, performance function f_1 is eliminated. In the yellow colored polytopes, performance function f_2 is eliminated.

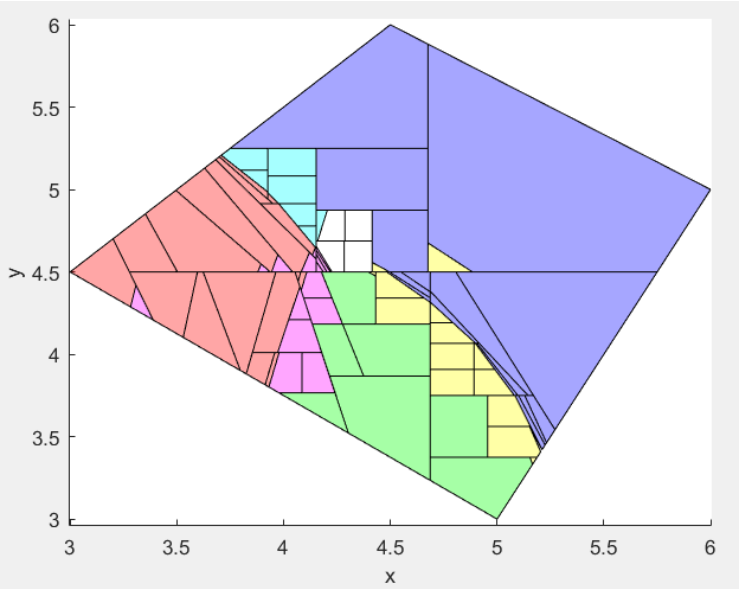

Figure 4.13: The total characterization after 150 seconds of execution of the polytope algorithm. Eliminated functions are blue: f_2, f_3, red: f_1, f_3, green: f_1, f_2, cyan: f_3, magenta: f_1 and yellow: f_2

4.2 Polytope Algorithm Pseudocode

In this section, we provide the pseudocode and its description for the polytope algorithm illustrated in Section 4.1.

In the remainder of this section, the word polytope does not only stand for the geometric specification but it rather represents an object with certain properties, its geometric specification is one of these properties. The properties of a polytope P can be listed as follows.

- $P.PS$ is the polytope specification that stands for the geometric specification of P.
- $P.FOU$ is the set of tuples $fou = (f, \overline{f}, \underline{f})$ where $fou.f$ is a potentially dominant performance function in $P.PS$, $fou.\overline{f}$ is an affine over-approximation of $fou.f$ over $P.PS$ and $fou.\underline{f}$ is an affine under-approximation of $fou.f$ over $P.PS$.
- $P.splitPars$ is the set of parameters whose domains have been split to create $P.PS$. We keep track of such parameters to avoid unnecessary calculation of approximations. That is, we only find new approximations for performance functions that have such parameters.
- $P.isApproximated$ is a binary variable showing if P is using approximations calculated over a parent polytope specification ($P_{parent}.PS \supset P.PS$) or approximations calculated over $P.PS$. It should be noted that an approximation calculated over $P_{parent}.PS$ is also an approximation over $P.PS$. However, an approximation calculated over $P.PS$ is generally more accurate than an approximation calculated over $P_{parent}.PS$.

Algorithm 1 solves the characterization problem for an input polytope parameter space given as a polytope specification *IPS* and a set of performance functions *IF*. As stated in the previous section, the partial derivatives of performance functions $f \in IF$ are assumed to be monotone with respect to all of their parameters. The characterization is terminated when the execution time reaches the input termination time *tt*. As illustrated in Section 4.1, Algorithm 1 first calculates affine over- and under-approximations of all performance functions $f \in IF$ over *IPS*. This is shown in Line 2 to Line 7. Given a polytope specification *PS*, functions and their approximations *FOU* and a set of parameters *Par*, Algorithm 2 finds new approximations for functions $fou.f \in FOU$ that have at least one parameter $par \in Par$. At the beginning of Algorithm 1, we need to find approximations for all functions $f \in IF$ over *IPS*. Therefore, in Line 7, the set of all parameters of *IPS* is provided as input to Algorithm 2. The input polytope is initialized in Line 9. Its polytope specification is equal to *IPS* considering that we want to characterize *IPS*, its *FOU* contains all the functions $f \in IF$ with their approximations found in Line 7, its *splitPars* is empty because this is the input polytope meaning that it has no parent polytope and its *isApproximated* is equal to *true* because we calculated the approximations of functions over its polytope specification $P.PS = IPS$. Then, in Line 11, this input polytope is pushed into *Queue*, which holds the polytopes to be characterized. The set *FCP*, in Line 13, contains the fully characterized polytopes.

Before continuing with Algorithm 1, we provide further details about Algorithm 2. In Examples 4.1.3 to 4.1.6, we illustrated how an approximation is calculated for a function. These steps are shown in Line 6 to Line 17 of Algorithm 2. First, in Line 6, we check if a parameter of *PS* is not included in $fou.f$. If this is true, then it means that we can use a lower-dimensional polytope specification *projectedPS* rather than using *PS*. Algorithm 3 shows how the polytope specification is projected to obtain a lower-dimensional polytope specification. As it was done for function f_3 in Example 4.1.6, we remove values of parameters (that are not included in f) from corners of polytope specification *PS*. If, in Line 6 of Algorithm 2, all the parameters of *PS* are included in $fou.f$, then there is no need for projection considering that we need to use the entire *PS*. In Line 10, we continue with finding a hyperrectangle *encapsulatingHyperrectangle* that encapsulates *projectedPS*. In Line 11, we create a uniform mesh *gridPoints* for *encapsulatingHyperrectangle* with the chosen *resolution* and calculate the corresponding *diameter* of mesh elements. The definition of the diameter of a polytope was given in Section 4.1 and an example of finding the diameter of mesh elements was given in Example 4.1.5. In Line 12, we find a polytope *encapsulatingPS* that encapsulates *projectedPS*. As stated in Example 4.1.3, using all the grid points $gp \in gridPoints$ to find an approximation of $fou.f$ over *PS* results in suboptimal approximations. Using only $gp \in gridPoints$ that are inside or on the border of *projectedPS* results in approximations that are not applicable for the entire *projectedPS*. This is the reason that we find this encapsulating polytope *encapsulatingPS* that contains the neighboring grid points of *projectedPS*. A detailed explanation about how this encapsulating polytope is calculated is given in *Lemma 2* of [32]. In Line 13, we create two sets of grid points $gridPoints_1$ and $gridPoints_2$. $gridPoints_1$ contains the grid points $gp \in gridPoints$ that are inside or on the border of *encapsulatingPS* while $gridPoints_2$ contains the grid points $gp \in gridPoints$ that are inside or on the border of *encapsulatingPS* but not inside of *projectedPS*. In Line 14, we solve the linear program 4.1 for $fou.f$ using $gridPoints_1$ to obtain affine approximations $\overline{A}, \overline{h}, \underline{A}, \underline{h}$ of $fou.f$ over *PS*. In Line 15, we find an approximation of approximation error σ by calculating the bound $\sigma_{bound} = \delta^* \max_{p \in PS} ||fou.f'(p)||_2 \geq \sigma$ using $gridPoints_2$ and *diameter*. In Lines 16 and 17,

we obtain affine over- and under-approximations of $fou.f$ by combining the obtained affine approximations with σ_{bound}.

Now, we return to Algorithm 1. In Line 14, we check the termination conditions. If *Queue* is empty, then it means that characterization of *IPS* is completed, resulting in termination of Algorithm 1. Algorithm 1 also terminates if the execution time exceeds the input termination time tt. If none of these terminating conditions are satisfied, then, in Line 15, we continue by taking the polytope at the head of *Queue* in order to be characterized. To characterize a polytope P, we first use Algorithm 4 to search for functions $fou.f \in P.FOU$ that are not dominant in the entire $P.PS$. As shown in Lines 2 to 4 of Algorithm 4, we search for function pairs $fou_i.f, fou_j.f$ that satisfy $fou_j.\underline{f}(c) > fou_i.\overline{f}(c)$ at all corners c of PS. We remove such $fou_i.f$ together with its approximations from FOU considering that $fou_i.f$ is proven to be not dominant in the entire PS. In Line 17 of Algorithm 1, we check if the size of $P.FOU$ is larger than one. If it is equal to one, then it means that the characterization of P is complete and, in Line 40, it is included in FCP. If the size is larger than one, then we continue with its characterization. In Line 18, we check if $P.PS$ has any parameter that is not included in any remaining $fou \in P.FOU$. We check for such parameters, because they are no longer needed for the characterization of the polytope P. Algorithm 5 shows how such parameters are removed from $P.PS$. Similar to Algorithm 3, we remove values of such parameters from the corners of PS and find the convex hull of these corner points. The resulting projection *projectedPS* has fewer corner points than P leading to fewer comparisons by Algorithms 4 and 6. In Line 9 of Algorithm 5, we also remove such parameters from *splitPars* considering that they are no longer included in PS.

In Line 20 of Algorithm 1, Algorithm 6 searches for remaining functions $fou.f \in P.FOU$ that are not dominant in part of $P.PS$. As shown in Lines 3 and 4 of Algorithm 6, we search for a function pair $fou_i, fou_j \in FOU$ that satisfies $fou_j.\underline{f}(c) > fou_i.\overline{f}(c)$ at any corner c of PS. If such a function pair is found, then it is proven that there is a hyperplane $fou_j.\underline{f} - fou_i.\overline{f}$ that splits PS into two smaller polytopes $PS_1 = \{p \in PS \,|\, fou_j.\underline{f}(p) - fou_i.\overline{f}(p) \geq 0\}$ and $PS_2 = \{p \in PS \,|\, fou_j.\underline{f}(p) - fou_i.\overline{f}(p) \leq 0\}$. Function $fou.i$ is not dominant in PS_1 while no elimination is done for PS_2. To create PS_1 and PS_2, we split the domains of parameters included in hyperplane $fou_j.\underline{f} - fou_i.\overline{f}$. Therefore, in Lines 9 and 10, such parameters are included in *splitPars*. In Line 21 of Algorithm 1, we check if such a hyperplane is found. If it is found, then we add the corresponding smaller polytopes P_1 and P_2 to *Queue* for further characterization.

If a hyperplane is not found in Line 20, then, in Line 27, we check if $P.isApproximated$ is false. If it is false, then it means that some of the approximations $fou.\overline{f}, fou.\underline{f} \in P.FOU$ can be suboptimal over $P.PS$ considering that all of these approximations are calculated over a parent polytope of $P.PS$. Therefore, in Line 28, we find new approximations calculated over $P.PS$. An important thing to note here is that, unlike in Line 7 of Algorithm 1, the input parameter set is $P.splitPars$. In Line 7, we needed to find approximations for all of the functions in IF. However, in Line 28, we only need to find new approximations to a function (calculated over $P.PS$) if these new approximations are expected to be better than the approximations at hand (which are approximations calculated over a parent polytope of $P.PS$). As shown in Lines 3 and 4 of Algorithm 2, if a function $fou.f$ does not have any

parameter $par \in splitPars$, then the domains of its parameters are not changed since the last calculation of its approximations $fou.\overline{f}$ and $fou.\underline{f}$ over a parent polytope of PS. If we calculate new approximations $\overline{f}_{new}$ and $\underline{f}_{new}$ of $fou.f$ over PS, then these approximations are going to be equal to the approximations $fou.\overline{f}$ and $fou.\underline{f}$. Therefore, we do not find new approximations for such functions. However, for the functions $fou.f$ which have at least one parameter $par \in splitPars$ we calculate new approximations. Because the domain of at least one parameter is reduced, the new approximations $\overline{f}_{new}$ and $\underline{f}_{new}$ are expected to be more accurate then the approximations $fou.\overline{f}$ and $fou.\underline{f}$. It should be noted that, in Line 29 of Algorithm 1, $P.splitPars$ is assigned to be empty because we need to keep track of changes in domains of parameters in between calculations of approximations. In Line 30, polytope P (with updated approximations) is added to *Queue* for further characterization.

If $P.isApproximated$ is true (Line 31), then the approximations $fou.\overline{f}$ and $fou.\underline{f}$ of $fou.f$ are calculated over $P.PS$ and we know that no further characterization is possible using these approximations. Therefore, to find better approximations and perform further characterization, we split $P.PS$ into two smaller polytopes and calculate new approximations over these smaller polytopes. Algorithm 7 shows how this splitting is done. As shown in Lines 3 to 8, we first find the hyperrectangle *encapsulatingHyperrectangle* that encapsulates PS. Then, we find the parameter *par* of *encapsulatingHyperrectangle* with the largest interval (domain) and the hyperplane that splits this interval into two equal intervals. Using this hyperplane, we split PS into two smaller polytopes PS_1 and PS_2. Because we only split the domain of *par* to create PS_1 and PS_2, that is the only parameter we add to $splitPars_{new}$ in Line 9. In Lines 33 and 34 of Algorithm 1, new approximations are calculated over PS_1 and PS_2. In Lines 37 and 38, polytopes P_1 and P_2 are added to *Queue* for further characterization.

When Algorithm 1 terminates, we return both *FCP* and *Queue*. Because *FCP* contains polytopes that are fully characterized while *Queue* contains polytopes that are partially characterized.

Algorithm 1 Solve the characterization problem for input polytope with specification *IPS* and input performance functions *IF*

1: **function** characterizePolytope(input polytope specification IPS, input performance function set IF, termination time tt)
2: $setOfAllPar = \{par \mid par$ is a parameter of $IPS\}$
3: *//Functions and their (over/under-)approximations (FOU) are represented as a set of tuples* $(f, \overline{f}, \underline{f})$
4: $FOU_{input} = \emptyset$
5: **for each** function $f \in IF$
6: $FOU_{input} = FOU_{input} \cup \{(f, -, -)\}$
7: $FOU_{input} = \text{findApproximations}(IPS, FOU_{input}, setOfAllPar)$
8: *//A polytope P is represented as a tuple* $P = (PS, FOU, splitPars, isApproximated)$
9: $P_{input} = (IPS, FOU_{input}, \emptyset, true)$
10: *// Queue containing the polytopes to be characterized*
11: $Queue$.enqueue(P_{input})
12: *// The set containing the fully characterized polytopes*
13: $FCP = \emptyset$
14: **while** $\neg Queue$.isEmpty && $executionTime < tt$
15: $P = Queue$.dequeue
16: $P.FOU = \text{removeNotDominantFunctions}(P.PS, P.FOU)$
17: **if** size$(P.FOU) > 1$ **then**
18: **if** a parameter of $P.PS$ is not included in any $fou \in P.FOU$
19: $P.PS, P.splitPars = \text{findProjectionToEliminateParameters}(P.PS, P.FOU, P.splitPars)$
20: $isHyperplaneFound, PS_1, PS_2, fou_{notDom}, splitPars =$ findHyperplaneToSplit$(P.PS, P.FOU, P.splitPars)$
21: **if** $isHyperplaneFound$ **then**
22: $P_1 = (PS_1, P.FOU \setminus \{fou_{notDom}\}, splitPars, false)$
23: $P_2 = (PS_2, P.FOU, splitPars, false)$
24: $Queue$.enqueue(P_1)
25: $Queue$.enqueue(P_2)
26: **else**
27: **if** $\neg P.isApproximated$ **then**
28: $FOU_{better} = \text{findApproximations}(P.PS, P.FOU, P.splitPars)$
29: $P = (P.PS, FOU_{better}, \emptyset, true)$
30: $Queue$.enqueue(P)
31: **else**
32: $PS_1, PS_2, splitPars = \text{split}(P.PS, P.splitPars)$
33: $FOU_{better_1} = \text{findApproximations}(PS_1, P.FOU, splitPars)$
34: $FOU_{better_2} = \text{findApproximations}(PS_2, P.FOU, splitPars)$
35: $P_1 = (PS_1, FOU_{better_1}, \emptyset, true)$
36: $P_2 = (PS_2, FOU_{better_2}, \emptyset, true)$
37: $Queue$.enqueue(P_1)
38: $Queue$.enqueue(P_2)
39: **else**
40: $FCP = FCP \cup \{P\}$
41: **end**
42: **return** $FCP, Queue$

Algorithm 2 Calculate affine over- and under-approximations of all performance functions $fou.f$, that include at least one parameter in *splitPars*, over polytope specification PS

function findApproximations(polytope specification PS, functions and their (over/under-)approximations FOU, parameter set *splitPars*)
  $FOU_{new} = \{\}$
  for each $fou.f$ that includes no parameter $par \in splitPars$, where $fou \in FOU$
    $FOU_{new} = FOU_{new} \cup \{fou\}$
  for each $fou.f$ that includes a parameter $par \in splitPars$, where $fou \in FOU$
    if a parameter of PS is not included in $fou.f$ **then**
      $projectedPS$ = findProjectionForFunction($fou.f, PS$)
    else
      $projectedPS = PS$
    $encapsulatingHyperrectangle$ = findEncapsulatingHyperrectangle($projectedPS$)
    $gridPoints, diameter$ = createUniformMesh($encapsulatingHyperrectangle, resolution$)
    $encapsulatingPS$ = findEncapsulatingPolytope($projectedPS, resolution$)
    $gridPoints_1, gridPoints_2$ = groupGridPoints($gridPoints, projectedPS, encapsulatingPS$)
    $\overline{A}, \overline{h}, \underline{A}, \underline{h}$ = findOverUnderLinearInterpolations($gridPoints_1, fou.f$)
    σ_{bound} = findApproximationErrorBound($fou.f, gridPoints_2, diameter$)
    $\overline{f}_{new} = \overline{A}\,p + \overline{h} + \sigma_{bound}$
    $\underline{f}_{new} = \underline{A}\,p + \underline{h} - \sigma_{bound}$
    $fou_{new} = (fou.f, \overline{f}_{new}, \underline{f}_{new})$
    $FOU_{new} = FOU_{new} \cup \{fou_{new}\}$
  return FOU_{new}

Algorithm 3 Find the projection of polytope specification PS that only includes parameters that are included in performance function f

function findProjectionForFunction(performance function f, polytope specification PS)
  $projectionOfCorners = \{c \mid c$ is a corner of $PS\}$
  for each parameter par of PS that is not included in f
    for each $c \in projectionOfCorners$
      // Point c becomes one less dimensional with each removal
      Remove value of par from c
  $projectedPS$ = Convex Hull of points $p \in projectionOfCorners$
  return $projectedPS$

Algorithm 4 Remove performance functions $fou.f$ that are not dominant in entire polytope specification PS

function removeNotDominantFunctions(polytope specification PS, functions and their (over/under)-approximations FOU)
  for each $(fou_i.\overline{f}, fou_j.\underline{f})$ where $fou_i, fou_j \in FOU$ with $i \neq j$
    if $fou_j.\underline{f}(c) > fou_i.\overline{f}(c)$ at all corners c of PS **then**
      $FOU = FOU \setminus \{fou_i\}$
  return FOU

Algorithm 5 Find the projection of polytope specification PS that does not include parameters that are not included in functions $fou.f$

1: **function** findProjectionToEliminateParameters(polytope specification PS, functions and their (over/under)-approximations FOU, parameter set Par)
2: $splitPars = Par$
3: $projectionOfCorners = \{c \mid c \text{ is a corner of } PS\}$
4: **for each** parameter par of PS that is not included in any $fou \in FOU$
5: **for each** $c \in projectionOfCorners$
6: *// Point c becomes one less dimensional with each removal*
7: Remove value of par from c
8: **if** $par \in splitPars$ **then**
9: $splitPars = splitPars \setminus \{par\}$
10: $projectedPS$ = Convex Hull of points $p \in projectionOfCorners$
11: **return** $projectedPS$

Algorithm 6 Using pairs of over- and under-approximations $(fou_i.\overline{f}, fou_j.\underline{f})$, find if there is a hyperplane $fou_j.\underline{f} - fou_i.\overline{f}$ that cuts polytope specification PS

1: **function** findHyperplaneToSplit(polytope specification PS, functions and their (over/under)-approximations FOU, set of parameters Par)
2: $splitPars = Par$
3: **for each** $(fou_i.\overline{f}, fou_j.\underline{f})$ where $fou_i, fou_j \in FOU$ with $i \neq j$
4: **if** $fou_j.\underline{f}(c) > fou_i.\overline{f}(c)$ at any corner c of PS **then**
5: $isHyperplaneFound = true$
6: $PS_1 = \{p \in PS \mid fou_j.\underline{f}(p) - fou_i.\overline{f}(p) \geq 0\}$
7: $PS_2 = \{p \in PS \mid fou_j.\underline{f}(p) - fou_i.\overline{f}(p) \leq 0\}$
8: $fou_{notDom} = fou_i$
9: **for each** parameter par that is included in $fou_j.\underline{f} - fou_i.\overline{f}$ and $par \notin splitPars$
10: $splitPars = splitPars \cup \{par\}$
11: **return** $isHyperplaneFound, PS_1, PS_2, fou_{notDom}, splitPars$
12: $isHyperplaneFound = false$
13: **return** $isHyperplaneFound, null$

Algorithm 7 Split polytope specification PS into two smaller polytope specifications

1: **function** split(polytope specification PS, set of parameters $splitPars$)
2: $splitPars_{new} = splitPars$
3: $encapsulatingHyperrectangle$ = findEncapsulatingHyperrectangle(PS)
4: *//The lower bound of a parameter par (lb_{par}) of a hyperrectangle H is its smallest value in H*
5: *//The upper bound of a parameter par (ub_{par}) of a hyperrectangle H is its largest value in H*
6: **for** the parameter par of $encapsulatingHyperrectangle$ with the largest interval $par_{int} = ub_{par} - lb_{par}$
7: $PS_1 = \{p \in PS \,|\, p_{par} \leq \frac{lb_{par}+ub_{par}}{2}\}$
8: $PS_2 = \{p \in PS \,|\, p_{par} \geq \frac{lb_{par}+ub_{par}}{2}\}$
9: $splitPars_{new} = splitPars_{new} \cup \{par\}$
10: **return** PS_1, PS_2, par

4.3 Conclusion

In this chapter, a polytope-based characterization algorithm is introduced to solve the previously defined characterization problem. For this algorithm, we assume that the given performance functions satisfy the same monotonicity properties as the third-order point-to-point motion profile. This algorithm calculates affine over- and under-approximation functions for each performance function by using these monotonicity properties to compare the values of the these performance functions. If the value of an under-approximation is greater than the value of an over-approximation at all the corner points of the characterized polytope, then the performance function with the corresponding over-approximation is eliminated from all the points in the characterized polytope considering that it can never be the maximum performance function.

A performance function can also be eliminated from part of the characterized polytope if the value of an under-approximation is greater than the value of its over-approximation at some corner points of the characterized polytope but not all. In this case, the characterized polytope is split into two smaller polytopes and the performance function with the corresponding over-approximation is only eliminated from one of these smaller polytopes. If the characterized polytope cannot be split as a result of performance function elimination, then it is divided manually by using the hyperplane that splits the encapsulating hyperrectangle into two equal-sized smaller hyperrectangles.

For the smaller polytopes (obtained either from the comparison of over- and under-approximations or from the manual splitting), we continue characterization by updating the affine over- and under-approximation functions and comparing these updated affine over- and under-approximation functions. These steps of updating the approximations, comparing them and splitting the characterized polytope is repeated until either we obtain the exact solution of the problem or the execution time of the algorithm exceeds the input termination time.

It is important to note that this algorithm does not guarantee the exact solution as a result.

It rather provides an approximation of the exact solution. As the input termination time increases, the approximation gets closer to the exact solution. For some characterization problems, it is possible to obtain the exact solution in a finite amount of time. However, this is not the case for all the characterization problems.

Chapter 5

Hyperrectangle-Based Characterization

In this chapter, we demonstrate a hyperrectangle-based characterization algorithm (hyperrectangle algorithm, for short) that solves the characterization problem introduced in Section 3.5. Similar to Chapter 4, in the first section, we solve the example characterization problem (given in Section 3.5) using the hyperrectangle algorithm. In the second section, we provide the pseudocode for the hyperrectangle algorithm with the further details.

We use two different hyperrectangle algorithms. One algorithm is the one presented in this chapter that uses a monotonicity assumption and the other one uses interval arithmetic. The reason that we use the algorithm with interval arithmetic is that the interval techniques are commonly used in state-of-the-art methods of constraint programming [41] and the generic applicability of interval arithmetic makes it possible to characterize performance functions with different properties. The reason that we use the algorithm with the monotonicity assumption is mainly motivated by the properties of the third-order point-to-point motion profiles. With the monotonicity assumption, we do not require interval techniques to find approximations of the maximum and minimum values of performance functions. We directly find the exact maximum and minimum values without using computationally expensive interval arithmetic. Comparing this algorithm with the polytope algorithm gives us an idea about how the polytope algorithm compares to a hyperrectangle-based algorithm that is also targeted towards the third-order point-to-point motion profiles.

5.1 Hyperrectangle Algorithm Example

The hyperrectangle algorithm is presented by solving the following characterization problem:

Find the mapping $C : \mathbb{H}_0 \to \wp(\mathcal{F}) \setminus \{\emptyset\}$ with $\mathcal{F} = \{f_1, f_2, f_3\}$ and $\mathbb{H}_0 = \{p \in \mathbb{R}^2 \mid Gp \geq g\}$ where

$$f_1(x, y) = x^3 + y^3 - 20x - 20y + 150$$
$$f_2(x, y) = -x^3 + y^2 + 20x + y + 110$$
$$f_3(x, y) = x^2 + 130$$

$$G = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \quad g = \begin{bmatrix} 3 \\ -6 \\ 3 \\ -6 \end{bmatrix}$$

This characterization problem is the one presented in Section 3.5. Unlike the polytope algorithm, the hyperrectangle algorithm only accepts hyperrectangle parameter spaces.

Example 5.1.1 Similar to Figures 4.1 and 4.2, Figure 5.1 shows performance functions f_1, f_2 and f_3 for the parameter space $\mathbb{H}_0$ and Figure 5.2 shows the solution of the characterization problem.

In Chapter 4, the polytope algorithm calculates affine approximations to compare performance functions and eliminate the ones that are not dominant. To make such comparisons, the hyperrectangle-based algorithms (the monotone version and the interval-arithmetic version) calculate bounds rather than affine approximations. For the hyperrectangle parameter space $\mathbb{H}_0$, these algorithms calculate bounds $[lb_{0,i}, ub_{0,i}]$ for all $f_i \in \mathcal{F}$ where $lb_{0,i} \leq f_i(p) \leq ub_{0,i}$ for all $p \in \mathbb{H}_0$. If a pair $f_i, f_j \in \mathcal{F}$ satisfies $lb_{0,j} > ub_{0,i}$, then it is proven that f_i is not dominant in $\mathbb{H}_0$.

For the monotone version of the hyperrectangle algorithm introduced in this chapter, we assume that the performance functions are monotone. With this assumption, we find the bound $[lb_{0,i}, ub_{0,i}]$ of a function $f_i \in \mathcal{F}$ by only using the corner points of $\mathbb{H}_0$. Because of the monotonicity assumption, the lower bound is equal to the minimum of the performance function $lb_{0,i} = \min_{p \in \mathbb{H}_0} f_i(p) = min_{0,i}$ and the upper bound is equal to the maximum of the performance function $ub_{0,i} = \max_{p \in \mathbb{H}_0} f_i(p) = max_{0,i}$. In the given characterization problem, performance functions f_1, f_2 and f_3 are monotone.

For the interval-arithmetic version of the hyperrectangle algorithm, we use interval arithmetic to find the bounds $[lb_{0,i}, ub_{0,i}]$ of a performance function $f_i \in \mathcal{F}$. This version does not require monotonicity and many of the standard arithmetic operations have a corresponding interval-arithmetic operation. Therefore, with this version, a more generic class of performance functions can be characterized compared to the monotone version, which can only characterize monotone performance functions. However, calculations using interval arithmetic are, in general, more computationally expensive and the calculated lower and upper bounds are, in general, not equal to the minimum and maximum of the performance functions. The details about interval arithmetic can be found in [26].

The first step of the monotone version of the hyperrectangle algorithm is to calculate the maximum and minimum values of all the performance functions in the entire parameter space.

Example 5.1.2 For the given characterization problem, the algorithm finds the maximum and minimum values of f_1, f_2 and f_3 in $\mathbb{H}_0$. We demonstrate this calculation by showing the steps for f_1. Performance function f_1 has both parameters (x and y) of $\mathbb{H}_0$. Therefore, we use the corners of the hyperrectangle $\mathbb{H}_{0,1} = \mathbb{H}_0$ to find the maximum ($max_{0,1}$) and minimum ($min_{0,1}$) values of f_1. The values of f_1 at the corner points of $\mathbb{H}_{0,1}$ are $f_1(3,3) = 84$, $f_1(3,6) = 213$, $f_1(6,3) = 213$ and $f_1(6,6) = 342$.

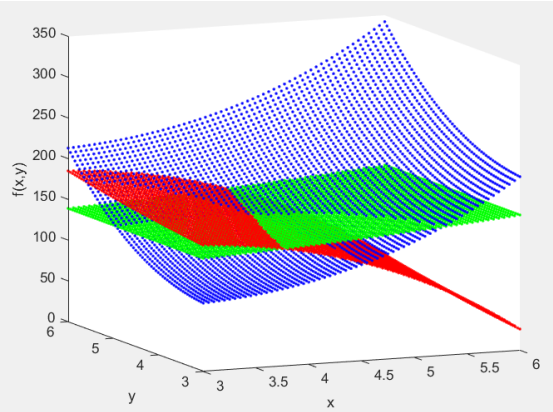

Figure 5.1: Sampled performance functions blue:$f_1(x, y)$, red:$f_2(x, y)$, green:$f_3(x, y)$

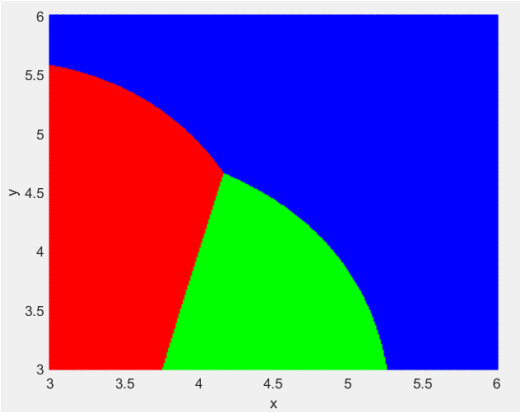

Figure 5.2: Sampled exact solution of the characterization problem blue:f_1, red:f_2, green:f_3

The maximum of f_1 is $max_{0,1} = max(f_1(3,3), f_1(3,6), f_1(6,3), f_1(6,6)) = 342$ and the minimum of f_1 is $min_{0,1} = min(f_1(3,3), f_1(3,6), f_1(6,3), f_1(6,6)) = 84$. Similar to f_1, performance function f_2 has both parameters of $\mathbb{H}_0$. Using all the corner points of $\mathbb{H}_0$, $min_{0,2} = min(f_2(3,3), f_2(3,6), f_2(6,3), f_2(6,6)) = 26$ and $max_{0,2} = max(f_2(3,3), f_2(3,6), f_2(6,3), f_2(6,6)) = 185$.

For performance functions that do not have all the parameters of the hyperrectangle, there is no need for using all of the corner points.

Example 5.1.3 In the given characterization problem, performance function f_3 only has one parameter (x). If we use all the corners of $\mathbb{H}_0$ to find the maximum and minimum values of f_3, then we would use redundant points where the value of f_3 is the same. This can be seen from $f_3(3,3) = f_3(3,6)$ and $f_3(6,3) = f_3(6,6)$. Therefore, similar to what we did in the polytope algorithm, we find the projection of the hyperrectangle parameter space (which has all the parameters) onto a lower dimensional subspace (which has only the parameters of the performance function whose maximum and minimum values are calculated) resulting in a hyperrectangle parameter space with fewer corners. f_3 only has parameter x and the minimum and maximum values of x in $\mathbb{H}_0$ are 3 and 6, respectively. Therefore, the projection is $\mathbb{H}_{0,3} = \{x \in \mathbb{R} \mid x \geq 3, x \leq 6\}$ with corner points $x = 3$ and $x = 6$. The maximum value of f_3 in $\mathbb{H}_0$ is $max_{0,3} = max(f_3(3), f_3(6)) = 166$ and $min_{0,3} = min(f_3(3), f_3(6)) = 139$.

We do the characterization of $\mathbb{H}_0$ by using the maximum and minimum values that we calculated for $f \in \mathcal{F}$. The set of potentially dominant performance functions in $\mathbb{H}_0$ is $\mathcal{F}_0 = \mathcal{F}$. As previously stated, we compare all the different pairs of maximum and minimum values $(max_{0,i}, min_{0,j})$ where $f_i, f_j \in \mathcal{F}_0$. If $min_{0,j} > max_{0,i}$ is satisfied for any $f_i, f_j \in \mathcal{F}_0$, then it is proven that performance function f_i is not dominant at any point in $\mathbb{H}_0$ resulting in $\mathcal{F}_0 := \mathcal{F}_0 \setminus \{f_i\}$. If the number of performance functions remaining in $\mathcal{F}_0$ is larger than one, then $\mathbb{H}_0$ is split into two smaller hyperrectangles. We repeat the same steps for these smaller hyperrectangles by calculating new maximum and minimum values for the remaining performance functions, eliminating the performance functions that are not dominant and

splitting the hyperrectangle if the number of remaining performance functions is more than one.

Example 5.1.4 There is no performance function pair $f_i, f_j \in \mathcal{F}_0$ that satisfies $min_{0,j} > max_{0,i}$. This is expected since there is no performance function $f \in \mathcal{F}$ that is not dominant at all $p \in \mathbb{H}_0$. Therefore, we split $\mathbb{H}_0$ into two equally-sized smaller hyperrectangles $\mathbb{H}_1 = \{p \in \mathbb{H}_0 \mid y \leq 4.5\}$ and $\mathbb{H}_1 = \{p \in \mathbb{H}_0 | y \geq 4.5\}$ with $\mathcal{F}_1 = \mathcal{F}_0$ and $\mathcal{F}_2 = \mathcal{F}_0$. For the characterization of $\mathbb{H}_1$, we calculate the maximum and minimum values of performance functions in $\mathbb{H}_1$. However, because only the interval $y \in [3, 6]$ is changed for the smaller hyperrectangles, we only need to update maximum and minimum values of performance functions that have y. Therefore, the maximum and minimum values of only f_1 and f_2 are calculated while the maximum and minimum values of f_3 stay the same resulting in $max_{1,1} = 247.125, min_{1,1} = 84, max_{1,2} = 167.75, min_{1,2} = 26, max_{1,3} = max_{0,3} = 166, min_{1,3} = min_{0,3} = 139$. Again, no performance function pair $f_i, f_j \in \mathcal{F}_1$ satisfies $min_{1,j} > max_{1,i}$. Therefore, no function is eliminated and we split $\mathbb{H}_1$ into two smaller hyperrectangles. This procedure described above is repeated for newly created smaller hyperrectangles until the algorithm terminates. Figure 5.3 shows the total characterization for $\mathbb{H}_0$ after 2 seconds of execution of the hyperrectangle algorithm. The blue, red and green colored hyperrectangles are completely characterized. The hyperrectangles that are not completely characterized are hard to see considering their sizes. Comparing Figures 4.13 and 5.3, we can see that, in a much lower amount of time, the hyperrectangle algorithm characterizes more regions than the polytope algorithm for the example problem.

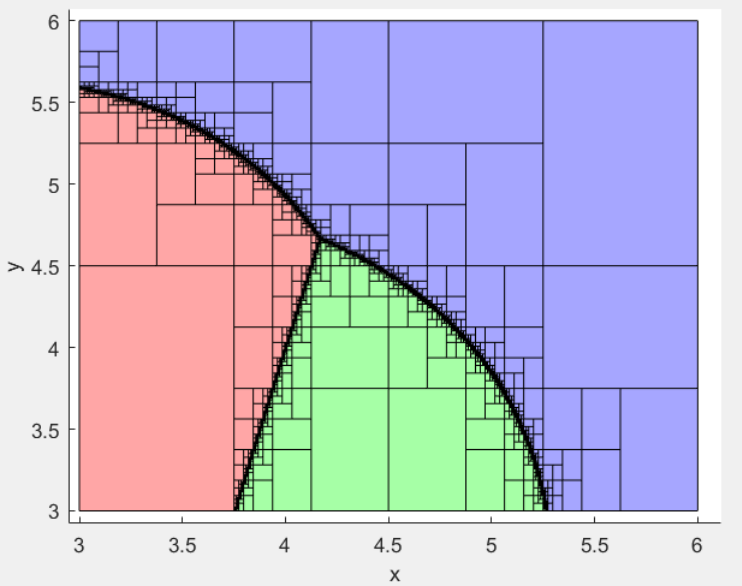

Figure 5.3: The total characterization after 2 seconds of execution of the hyperrectangle algorithm

5.2 Hyperrectangle Algorithm Pseudocode

In this section, we provide the pseudocode and its description for the hyperrectangle algorithm illustrated in Section 5.1.

Similar to Section 4.2, in the remainder of this section, the word hyperrectangle does not only stand for the geometric specification but it rather represents an object with certain properties. The properties of a hyperrectangle H can be listed as follows.

- $H.HS$ is the hyperrectangle specification which stands for the geometric specification of H.
- $H.FMM$ is the set of tuples $fmm = (f, max, min)$ where $fmm.f$ is a potentially dominant performance function in $H.HS$, $fmm.max$ is the maximum value of $fmm.f(p)$ for all $p \in H.HS$ and $fmm.min$ is the minimum value of $fmm.f(p)$ for all $p \in H.HS$.

Algorithm 8 solves the characterization problem for a hyperrectangle parameter space with hyperrectangle specification IHS and a set of performance functions IF. As stated in the previous section, the performance functions $f \in IF$ are assumed to be monotone. The characterization is terminated when the execution time exceeds the input termination time tt. Algorithm 8 first calculates the maximum and minimum values of all performance functions $f \in IF$ in IHS. This is shown in Lines 4 to 7. As illustrated in Section 5.1, we find these maximum and minimum values by checking the values of $f \in IF$ at the corner points of IHS. The input hyperrectangle is initialized at Line 9. Its hyperrectangle specification is equal to IHS and its FMM contains all the functions $f \in IF$ with their maximum and minimum values in IHS. Then, at Line 11, this input hyperrectangle is pushed into $Queue$ which holds the hyperrectangles to be characterized. The set FCH, at Line 13, contains the fully characterized hyperrectangles.

Similar to Algorithm 1, Algorithm 8 terminates if $Queue$ is empty or the execution time exceeds the input termination time tt. If these conditions are not satisfied, then, at Line 15, we continue characterization by taking the hyperrectangle at the head of $Queue$ in order to be characterized. At Line 16, Algorithm 9 searches for functions $fmm.f$ that are not dominant in the entire $H.HS$. As shown in Lines 2 to 4 of Algorithm 9, we search for function pairs $fmm_i.f, fmm_j.f$ that satisfy $fmm_j.min > fmm_i.max$. We remove such fmm_i from FMM as $fmm_i.f$ is proven to be not dominant in entire $H.HS$. At Line 17 of Algorithm 8, we check if the characterization of H is complete or not. If it is complete, then, at Line 28, it is included in FCH. If it is not complete, then, at Line 18, Algorithm 10 splits $H.HS$ into two equally sized smaller hyperrectangles.

As shown at Lines 4 to 6 of Algorithm 10, HS is split by splitting the domain that is the largest among all its parameters into two. At Line 7, the parameter with this largest domain is stored in $splitPar$. At Lines 19 and 20 of Algorithm 8, we find new maximum and minimum values for functions (that are remaining in $H.FMM$) whose maximum and minimum values are changed due to the splitting of $H.HS$ into HS_1 and HS_2. As shown at Lines 3 to 6 of Algorithm 11, we only calculate new maximum and minimum values for functions $fmm.f$ that have $splitPar$, because only the domain of $splitPar$ is different in HS_1 and HS_2 from $H.HS$. For the remaining functions, at Lines 7 and 8, we use the

already calculated maximum and minimum values. At Lines 21 and 22 of Algorithm 8, the smaller hyperrectangles H_1 and H_2 are initialized with the corresponding hyperrectangle specifications and, maximum and minimum values. Then, at Lines 23 and 24, they are added to *Queue* for further characterization. Similar to Algorithm 1, when Algorithm 8 terminates, we return both *FCH* and *Queue*.

Algorithm 8 Solve the characterization problem for input hyperrectangle with specification *IHS* and input performance functions *IF*

1: **function** characterizeHyperrectangle(input hyperrectangle specification *IHS*, input performance function set *IF*, termination time *tt*)
2: *// Functions and their maximum and minimum values (FMM) are represented as a set of tuples* (f, max, min)
3: $FMM_{input} = \{\}$
4: **for each** function $f \in IF$
5: $max = \max_{p \in IHS} f(p)$
6: $min = \min_{p \in IHS} f(p)$
7: $FMM_{input} = FMM_{input} \cup \{(f, max, min)\}$
8: *// A hyperrectangle H is represented as a tuple* $H = (HS, FMM)$
9: $H_{input} = (IHS, FMM_{input})$
10: *// Queue containing the hyperrectangles to be characterized*
11: $Queue$.enqueue(H_{input})
12: *// The set containing the fully characterized hyperrectangles*
13: $FCH = \{\}$
14: **while** $\neg Queue$.isEmpty && $executionTime < tt$
15: $H = Queue$.dequeue
16: $H.FMM$ = removeNotDominantFunctions($H.FMM$)
17: **if** size($H.FMM$) > 1 **then**
18: $HS_1, HS_2, splitPar$ = splitInTwo($H.HS$)
19: FMM_1 = findMinMax($HS_1, H.FMM, splitPar$)
20: FMM_2 = findMinMax($HS_2, H.FMM, splitPar$)
21: $H_1 = (HS_1, FMM_1)$
22: $H_2 = (HS_2, FMM_2)$
23: $Queue$.enqueue(H_1)
24: $Queue$.enqueue(H_2)
25: **else**
26: $FCH = FCH \cup \{H\}$
27: **return** FCH, $Queue$

Algorithm 9 Remove performance functions $fmm.f$ that are not dominant in the entire hyperrectangle

function removeNotDominantFunctions(functions and their maximum and minimum values FMM)
  for each $(fmm_i.max, fmm_j.min)$ where $fmm_i, fmm_j \in FMM$ and $i \neq j$
    if $fmm_j.min > fmm_i.max$
      $FMM = FMM \setminus \{fmm_i\}$
  return FMM

Algorithm 10 Split the hyperrectangle specification HS into two smaller hyperrectangle specifications HS_1 and HS_2

function splitInTwo(hyperrectangle specification HS)
  //The lower bound of a parameter par (lb_{par}) is its smallest value in HS
  //The upper bound of a parameter par (ub_{par}) is its largest value in HS
  for the parameter par of HS with the largest interval $par_{int} = ub_{par} - lb_{par}$
    $HS_1 = \{p \in HS \mid p_{par} \leq \frac{lb_{par}+ub_{par}}{2}\}$
    $HS_2 = \{p \in HS \mid p_{par} \geq \frac{lb_{par}+ub_{par}}{2}\}$
    $splitPar = par$
  return $HS_1, HS_2, splitPar$

Algorithm 11 Find the minimum and maximum of functions $fmm.f$ that have $splitPar$ as a parameter in hyperrectangle specification HS

function findMinMax(hyperrectangle specification HS, functions and their maximum and minimum values FMM, parameter $splitPar$)
  $FMM_{new} = \{\}$
  for each function $fmm.f$ that includes $splitPar$ where $fmm \in FMM$
    $max_{new} = \max_{p \in HS} fmm.f(p)$
    $min_{new} = \min_{p \in HS} fmm.f(p)$
    $FMM_{new} = \{(fmm.f, max_{new}, min_{new})\}$
  for each function $fmm.f$ that does not include par where $fmm \in FMM$
    $FMM_{new} = FMM_{new} \cup \{fmm\}$
  return FMM_{new}

5.3 Conclusion

In this chapter, two hyperrectangle-based characterization algorithms are introduced to solve the previously defined characterization problem. One of these algorithms is presented by solving an example characterization problem and giving the pseudocode. For this presented algorithm, it is assumed that the given performance functions satisfy the same monotonicity properties as the third-order point-to-point motion profile. These hyperrectangle-based characterization algorithms uses upper and lower bounds to compare the values of performance functions. For the presented algorithm, we use the monotonicity properties to find the maximum and minimum values of each performance function in a hyperrectangular parameter space and these maximum and minimum values are used as the upper and lower bounds of the performance functions. The only difference between the hyperrectangle-based characterization algorithms is the way that they calculate the upper and lower bounds. The presented algorithm uses the monotonicity assumption while the other algorithm uses interval arithmetic which does not require such assumptions.

If the lower bound of a performance function is greater than the upper bound of another performance function, then the value of the performance function with the corresponding upper bound can never be maximum at any point in the characterized hyperrectangular parameter space. We eliminate such performance functions and if the number of remaining performance functions is larger than one, then the characterization is not complete. If the characterization is not complete and the execution time of the algorithm is less than the input termination time, then the hyperrectangular parameter space is divided into two equal-sized hyperrectangles. The previous steps of finding upper and lower bounds, comparing them and splitting the characterized hyperrectangular parameter space is repeated until either we obtain the exact solution of the problem or the execution time of the algorithm exceeds the input termination time.

Just like the polytope-based algorithm, the hyperrectangle-based algorithms also provide approximations of the exact solution as a result.

Chapter 6

Performance Analysis of the Characterization Algorithms

In this chapter, we analyse the performance of the polytope- and hyperrectangle-based characterization algorithms, presented in Chapters 4 and 5, by providing the results of our experiments and the corresponding conclusions. Our main focus is on the analysis of the polytope algorithm and the monotone version of the hyperrectangle algorithm. Because both of these algorithms were targeted towards the third-order point-to-point motion profiles used by LSAT. However, for general applicability, we also implemented the interval-arithmetic version of the hyperrectangle algorithm.

The polytope algorithm and the monotone hyperrectangle algorithm are implemented in the C language using the cddlib library [15]. This library uses the double description method [16] for the representation of polytopes and the operations are done using the GMP library [20]. For correctness of the algorithms, we use exact arithmetic for both of the algorithms. The interval-arithmetic version of the hyperrectangle algorithm is implemented in the C++ language using the IBEX library. All of the algorithms are run on a 64-bit Ubuntu machine.

6.1 Characteristics of the Algorithms

The polytope- and hyperrectangle-based algorithms have two main parts: the calculation of affine approximations/bounds and the characterization done by using these affine approximations/bounds. In the monotone version of the hyperrectangle algorithm, the bounds of a performance function are simply calculated by checking the function values at the corners of hyperrectangles. In the polytope algorithm, the calculation of an affine approximation includes the formation of a mesh and the solution of the LP given in Equation 4.1. Therefore, the calculation of affine approximations is more computationally expensive than the calculation of bounds.

In the hyperrectangle algorithm, the characterization of a hyperrectangle is done by using the calculated bounds. By comparing the lower and upper bounds of different performance functions, we eliminate performance functions that are not dominant in the entire hyperrectangle. In the polytope algorithm, the characterization of a polytope is done by using the

calculated affine approximations. By comparing the affine over- and under-approximations of different performance functions, we either eliminate performance functions that are not dominant in the entire polytope or we characterize part of the polytope by splitting it into two smaller polytopes and eliminating the performance function in one of these smaller polytopes.

Intuitively, the amount of characterization resulting from comparing affine approximations is expected to be greater than the amount of characterization resulting from comparing bounds, because affine approximations can eliminate functions either in the entire polytopes or part of the polytopes while bounds can only eliminate functions in the entire hyperrectangles. Then, our aim is to answer the question: *Is it possible for the polytope algorithm to characterize more area/volume per time unit than the hyperrectangle algorithm by using computationally expensive affine approximations that, potentially, provide better characterization compared to bounds?*

6.2 Experiments and Results

The main factors that affect the computation time of the calculation of a performance function's affine approximations in the polytope algorithm are the following:

- The number of parameters that the performance function has ($parNum$)
- The chosen resolution (res)

For a performance function f with $parNum_f$ parameters and a chosen resolution res, the number of grid points in the mesh is equal to res^{parNum_f}, meaning that the computation of the affine approximations is done in exponential time. Therefore, the scalability of the polytope algorithm is mainly limited to performance functions with only a few parameters. The chosen resolution introduces a trade-off. If a low resolution is chosen, then the accuracy of the affine approximations is expected to be low. This means that the computation time of the affine approximations is low, but the regions that are characterized using these affine approximations are expected to be small. If a high resolution is chosen, then the accuracy of the affine approximations is expected to be higher. In general, this results in better characterization, meaning that the total number of affine approximations that need to be calculated is lower than with low resolution. In our experiments, we analyze this trade-off by testing with different resolutions.

We quantify the characterization results by calculating the volume of the characterized regions. Calculation of the volume of an n-dimensional polytope is a complex task [6]. Considering that we were able to compute the exact volumes of polytopes with three or fewer dimensions, the parameter spaces in the experiments have three or fewer dimensions.

Tables 6.1 to 6.6 and 6.11 to 6.13 provide the results of experiments with generic performance functions. We use these experiments to analyse the scalability and performance of the algorithms. Tables 6.7 to 6.10 provide the results of experiments with third-order point-to-point motion profiles.

For the polytope algorithm, the first column in the results tables shows the chosen resolution. The second column shows the percentage of characterized area of the total area.

The third column shows the chosen termination time. The fourth column shows the average time needed to calculate affine approximations of a performance function. The last column shows the total number of calculated affine approximations for the performance functions. For each resolution, the first row shows the results for the case in which we calculate affine approximations only once for each performance function. This provides us with a base value for the average calculation time of affine approximations.

Table 6.1 shows the results of the polytope algorithm for the characterization problem with two performance functions $f_1 = x^2, f_2 = y^2$ and parameter space $x \in [1, 2], y \in [1, 2]$. It can be seen that the average calculation time of affine approximations increases with the increasing total number of calculated affine approximations. This is a result of the GMP rational numbers that we use for exact arithmetic. GMP represents a rational number using a denominator and a numerator. The computation time of operations on rational numbers increases with increasing values of the denominator and numerator of the rational numbers. With an increasing number of iterations of the polytope algorithm, the parameter space is divided into more and more polytopes. These divisions introduce more and more factors into the denominators and numerators, which results in increased values. For example, for resolution equal to 10, some of the grid points that are used for the calculation of affine approximations over the initial polytope are $1000/999, 1001/999$ and $334/333$. Some of the grid points that are used for the seventh affine approximation calculation are (14080...57291)/(99484...23168) and (70360...11377)/(49692...72416). We do not include the entire numbers because they have approximately 350 digits. This explains the increase in the average calculation time of affine approximations as the total number of calculated affine approximations increases with increasing termination time.

In Table 6.1, we can see that, for the same termination time, the percentage of characterized area increases with an increasing resolution up to some point. For example, the percentage of characterized area for resolution 10 and termination time 5 seconds is 0.98171. For resolution 50 and termination time 5, this increases to 0.99252. However, starting from resolution 100, we can see that the percentage of characterized area decreases for the same termination time with increasing resolution. This is the result of the aforementioned trade-off between the accuracy of the affine approximations and the time it takes to calculate them.

Table 6.2 shows the results of the monotone hyperrectangle algorithm for the same characterization problem. We can see that, unlike the polytope algorithm, the average calculation time of a bound does not increase with the increasing total number of calculated bounds. Similar to the polytope algorithm, the monotone hyperrectangle algorithm divides the parameter space into more and more hyperrectangles as the number of iterations increases. However, unlike the polytope algorithm, the hyperrectangles are always divided down the middle. Therefore, this division only introduces a factor of 2 to the denominator of some of the corners. This does not result in a significant change in the computation time of rational number operations. Another reason is that the calculation of a bound is done by checking the function values at the corners of the hyperrectangles. Therefore, the number of performed operations is much more lower than the number of operations that is needed to calculate affine approximations.

Comparing the results in Tables 6.1 and 6.2, we see that, for these performance func-

tions and parameter space, the monotone hyperrectangle algorithm always characterizes more area per second than the polytope algorithm. It should be noted that we do not provide the percentage of characterized area of the monotone hyperrectangle algorithm for high termination times, because it is already clear from the lower termination times that the monotone hyperrectangle algorithm performs better.

Resolution	Percentage of Characterized Area	Termination Time (seconds)	Average Calculation Time of Approximations (seconds)	Number of Calculated Approximations
10	0.59220	0.009438	0.001208	2
10	0.95712	1	0.006876	78
10	0.98171	5	0.015201	188
10	0.98808	10	0.019771	282
50	0.79119	0.012779	0.002907	2
50	0.98052	1	0.015938	54
50	0.99252	5	0.038108	114
50	0.99572	10	0.051742	164
100	0.81578	0.016817	0.004818	2
100	0.97643	1	0.021415	42
100	0.99179	5	0.055885	84
100	0.99578	10	0.077118	124
200	0.82804	0.024967	0.008946	2
200	0.97080	1	0.029087	34
200	0.99114	5	0.093018	66
200	0.99288	10	0.130452	76
300	0.83212	0.034175	0.013344	2
300	0.96964	1	0.032124	30
300	0.98917	5	0.127365	54
300	0.99189	10	0.154221	66

Table 6.1: Polytope algorithm: characterization results for $f_1 = x^2, f_2 = y^2$ with $x \in [1, 2], y \in [1, 2]$

Percentage of Characterized Area	Termination Time (seconds)	Average Calculation Time of a Bound (seconds)	Number of Calculated Bounds
0.97875	0.005	0.00000295	786
0.99023	0.01	0.00000291	1518
0.99902	0.1	0.00000302	15066
-	5	0.00000298	756254
-	10	0.00000296	1581102

Table 6.2: Monotone hyperrectangle algorithm: characterization results for $f_1 = x^2, f_2 = y^2$ with $x \in [1, 2], y \in [1, 2]$

Table 6.3 shows the results of the polytope algorithm for the characterization problem with two performance functions $f_1 = x^4, f_2 = y^4$ and parameter space $x \in [1, 2], y \in [1, 2]$. It should be noted that, even if the performance functions are slightly changed compared to the previous experiment, the solution of the previous characterization problem and this characterization problem are the same. In Table 6.3, similar to Table 6.1, we can see that the average calculation time of approximations increases with the increasing resolution and termination time. Compared to Table 6.1, for the same resolution and termination time, we can see that the percentage of characterized area decreases. For example, for resolution 100 and termination time 10, the percentage of characterized area is 0.98539 in Table 6.3, while, for the same resolution and termination time, the percentage of characterized area is 0.99578 in Table 6.1. This is expected because the first-order partial derivatives of the performance functions $f_1 = x^4, f_2 = y^4$ are increasing faster than the first-order partial derivatives of $f_1 = x^2, f_2 = y^2$. In other words, the calculated affine approximations of x^4 are less accurate than the calculated affine approximations of x^2. Another reason for this decrease in the percentage of characterized area is the increase in the average calculation time of approximations, which results in fewer calculated approximations for the same termination time. We can see that, in Table 6.3, the average calculation time of approximations increased for the same resolution and termination time compared to Table 6.1. For example, for resolution 10 and termination time 5, the average calculation time of approximations is equal to 0.021221 in Table 6.3, while, for the same resolution and termination time, the average calculation time of approximations is equal to 0.015201 in Table 6.1, because the number of operations that is needed to calculate affine approximations of $f_1 = x^4, f_2 = y^4$ is greater than the number of operations needed for $f_1 = x^2, f_2 = y^2$. We can see that this increase in the average calculation time of approximations results in fewer calculated approximations for the same resolution and termination time in Table 6.3 compared to Table 6.1.

Table 6.4 shows the results of the monotone hyperrectangle algorithm for the same characterization problem in Table 6.3. Similar to the results for the polytope algorithm, we can see that the average calculation time of a bound for the same termination time increased in Table 6.4 compared to Table 6.2. This is again caused by the increased number of operations needed for x^4 compared to x^2. However, the increase in the average calculation time of a bound from Table 6.2 to Table 6.4 is approximately 5% while the increase in the average calculation time of approximations from Table 6.1 to Table 6.3 is approximately

50% when the resolution is equal to 10. This percentage even increases more with increasing resolution. Therefore, the increase in the number of operations, which is needed to calculate performance functions, increases the average calculation time of approximations much more significantly than the average calculation time of a bound.

Comparing Tables 6.4 and 6.3, we can see that, for these performance functions and parameter space, the monotone hyperrectangle algorithm always characterizes more area per second than the polytope algorithm, similar to the previous experiment.

Resolution	Percentage of Characterized Area	Termination Time (seconds)	Average Calculation Time of Approximations (seconds)	Number of Calculated Approximations
10	0.28205	0.009206	0.001210	2
10	0.92708	1	0.009153	66
10	0.96992	5	0.021221	158
10	0.98145	10	0.028093	226
50	0.52503	0.012351	0.002775	2
50	0.91516	1	0.023945	40
50	0.97479	5	0.060637	76
50	0.98460	10	0.085490	110
100	0.55737	0.016777	0.004924	2
100	0.92685	1	0.024880	36
100	0.96938	5	0.075992	64
100	0.98539	10	0.111001	86
200	0.57365	0.024613	0.008802	2
200	0.91647	1	0.037113	30
200	0.96426	5	0.100251	50
200	0.97252	10	0.163213	62
300	0.57910	0.033455	0.013199	2
300	0.91868	1	0.038040	28
300	0.93876	5	0.114599	44
300	0.96564	10	0.192580	54

Table 6.3: Polytope algorithm: characterization results for $f_1 = x^4, f_2 = y^4$ with $x \in [1, 2], y \in [1, 2]$

Percentage of Characterized Area	Termination Time (seconds)	Average Calculation Time of a Bound (seconds)	Number of Calculated Bounds
0.97705	0.005	0.00000308	790
0.98950	0.01	0.00000307	1562
0.99902	0.1	0.00000308	14850
-	5	0.00000323	744258
-	10	0.00000326	1469698

Table 6.4: Monotone hyperrectangle algorithm: characterization results for $f_1 = x^4, f_2 = y^4$ with $x \in [1, 2], y \in [1, 2]$

Table 6.5 shows the results of the polytope algorithm for the characterization problem with two performance functions $f_1 = x^3 + y^2$, $f_2 = x^2 + y^3$ and parameter space $x \in [1, 2], y \in [1, 2]$. The solution of this characterization problem is the same as the solutions of the previous characterization problems. The major difference between the previous experiments and this one is the number of parameters of the performance functions. The performance functions of this characterization problem have two parameters while the performance functions in the previous characterization problems have one parameter. We can see that this results in a significant increase in the average calculation time of approximations in Table 6.5 compared to Tables 6.1 and 6.3. For example, for resolution 100 and termination time 10, the average calculation time of approximations is equal to 2.046528 in Table 6.5, while, for the same resolution and termination time, it is equal to 0.077118 and 0.111001 in Tables 6.1 and 6.3, respectively. This is expected because the number of grid points increases exponentially with the increasing number of function parameters. This increase in the average calculation time of approximations decreases the total number of calculated approximations for the same termination time, resulting in a decreased percentage of characterized area. For example, for resolution 100 and termination time 10, the number of calculated approximations is equal to 10 and the percentage of characterized area is equal to 0.60853 in Table 6.5. For the same resolution and termination time, the number of calculated approximations is equal to 124 and the percentage of characterized area is equal to 0.99578 in Table 6.1 and the number of calculated approximations is equal to 86 and the percentage of characterized area is equal to 0.98539 in Table 6.3.

Table 6.6 shows the results of the monotone hyperrectangle algorithm for the same characterization problem. Similar to the polytope algorithm, there is a significant increase in the average calculation time of a bound in Table 6.6 compared to Tables 6.2 and 6.4. This is caused by the increase in the number of corner points that is needed to be checked for each function resulting from the increase in the number of parameters. For example, for termination time 0.01, the average calculation time of a bound is 0.00001083 in Table 6.6, while, for the same termination time, the average calculation time of a bound is 0.00000291 in Table 6.2 and 0.00000307 in Table 6.4. This is an approximately 300% increase in the average calculation time of a bound. For the polytope algorithm, the percentage of increase in the average calculation time of approximations is approximately 2000% from Tables 6.1 and 6.3 to Table 6.5. Also, with the increasing termination time, there is no significant change in the average calculation time of a bound for the monotone hyperrectangle algorithm. For termination time 0.005, the average calculation time of a bound is equal to 0.00001059 and, for termination time 10, the average calculation time of a bound is equal to 0.00001115, which is approximately a 5% increase. For the polytope algorithm, there is a much more substantial increase in the average calculation time of approximations with increasing termination time. In Table 6.5, for resolution 100 and termination time 5, the average calculation time of approximations is equal to 1.158823. For the same resolution with termination time 50, the average calculation time of approximations is equal to 3.573191 which is approximately a 300% increase.

Similar to the previous experiments, comparing Tables 6.5 and 6.6, we can see that the monotone hyperrectangle algorithm always characterizes more area per second than the polytope algorithm.

Resolution	Percentage of Characterized Area	Termination Time (seconds)	Average Calculation Time of Approximations (seconds)	Number of Calculated Approximations
10	0.00693	0.025529	0.009421	2
10	0.51752	1	0.022098	44
10	0.79952	5	0.042229	112
10	0.86495	10	0.056092	168
10	0.94996	50	0.102464	450
50	0.30254	0.404440	0.198874	2
50	0.37379	1	0.312391	6
50	0.52652	5	0.628326	12
50	0.60740	10	0.787105	16
50	0.83943	50	1.476834	34
100	0.36389	1.81527	0.904168	2
100	0.44198	5	1.158823	6
100	0.60853	10	2.046528	10
100	0.73011	50	3.573191	18
200	0.39619	8.918722	4.455851	2
200	0.64761	50	7.040665	10
200	0.64761	100	9.822113	12
200	0.67985	200	14.782102	14
300	0.40719	23.955888	11.9743	2
300	0.48749	50	12.610881	6
300	0.50539	100	14.237742	8
300	0.66045	200	22.561653	12

Table 6.5: Polytope algorithm: characterization results for $f_1 = x^3 + y^2, f_2 = x^2 + y^3$ with $x \in [1, 2], y \in [1, 2]$

Percentage of Characterized Area	Termination Time (seconds)	Average Calculation Time of a Bound (seconds)	Number of Calculated Bounds
0.45312	0.005	0.00001059	370
0.64843	0.01	0.00001083	718
0.94287	0.1	0.00001087	7082
-	5	0.00001111	332406
-	10	0.00001115	686946

Table 6.6: Monotone hyperrectangle algorithm: characterization results for $f_1 = x^3 + y^2, f_2 = x^2 + y^3$ with $x \in [1, 2], y \in [1, 2]$

Tables 6.7 to 6.10 show the results of the experiments with the third-order point-to-point motion profiles. It can be seen that the monotone hyperrectangle algorithm always characterizes more area per second than the polytope algorithm in these experiments too. We conclude that, in all of the experiments, the monotone hyperrectangle algorithm always performs better than the polytope algorithm.

Resolution	Percentage of Characterized Area	Termination Time (seconds)
10	0.94488	1
10	0.97884	5
10	0.98720	10
10	0.99587	50
50	0.97242	1
50	0.99283	5
50	0.99599	10
50	0.99854	50
100	0.97004	1
100	0.99248	5
100	0.99536	10
100	0.99853	50
200	0.97052	1
200	0.97913	5
200	0.98911	10
200	0.99783	50
300	0.94623	1
300	0.97552	5
300	0.98741	10
300	0.99700	50

Table 6.7: Polytope algorithm: characterization results for $f_1 = f_{mp_1}(50, 20, aMax_1, 4), f_2 = f_{mp_2}(50, 20, aMax_2, 4)$ with $aMax_1 \in [4, 5], aMax_2 \in [4, 5]$

Percentage of Characterized Area	Termination Time (seconds)
0.96875	0.01
0.99660	0.1
0.99975	1

Table 6.8: Monotone hyperrectangle algorithm: characterization results for $f_1 = f_{mp_1}(50, 20, aMax_1, 4), f_2 = f_{mp_2}(50, 20, aMax_2, 4)$ with $aMax_1 \in [4, 5], aMax_2 \in [4, 5]$

Resolution	Percentage of Characterized Volume	Termination Time (seconds)
10	0.67098	1
10	0.83524	5
10	0.87196	10
10	0.93001	50
50	0.76850	1
50	0.79687	5
50	0.81768	10
50	0.92476	50
100	0.35376	1
100	0.80002	5
100	0.80002	10
100	0.86635	50
200	0.36183	1
200	0.78855	5
200	0.78855	10
200	0.81551	50
300	0.36456	1
300	0.36456	5
300	0.36456	10
300	0.82072	50

Table 6.9: Polytope algorithm: characterization results for $f_1 = f_{mp_1}(40, vMax_1, 2.5, 4)$, $f_2 = f_{mp_2}(40, vMax_2, aMax_2, 4)$ with $vMax_1 \in [4, 5]$, $vMax_2 \in [4, 5]$, $aMax_2 = [2, 3]$

Percentage of Characterized Volume	Termination Time (seconds)
0.70312	0.01
0.87670	0.1
0.96308	1

Table 6.10: Monotone hyperrectangle algorithm: characterization results for $f_1 = f_{mp_1}(40, vMax_1, 2.5, 4)$, $f_2 = f_{mp_2}(40, vMax_2, aMax_2, 4)$ with $vMax_1 \in [4, 5]$, $vMax_2 \in [4, 5]$, $aMax_2 = [2, 3]$

Tables 6.11, 6.12 and 6.13 show the results of the interval-arithmetic version of the hyperrectangle algorithm for the same experiments given in Tables 6.1 to 6.6. In all of the experiments, the interval-arithmetic hyperrectangle algorithm always characterizes more area per second than the polytope algorithm and it performs slightly worse than the monotone hyperrectangle algorithm. For example, for termination time 0.1, the percentage of characterized area is equal to 0.99902 in Table 6.2 and 0.99721 in Table 6.11. This slight decrease in performance may be acceptable because of its general applicability compared to the monotone hyperrectangle algorithm.

Percentage of Characterized Area	Termination Time (seconds)
0.95117	0.005
0.97705	0.01
0.99721	0.1

Table 6.11: Interval-arithmetic hyperrectangle algorithm: characterization results for $f_1 = x^2, f_2 = y^2$ with $x \in [1,2], y \in [1,2]$

Percentage of Characterized Area	Termination Time (seconds)
0.95703	0.005
0.97851	0.01
0.99804	0.1

Table 6.12: Interval-arithmetic hyperrectangle algorithm: characterization results for $f_1 = x^4, f_2 = y^4$ with $x \in [1,2], y \in [1,2]$

Percentage of Characterized Area	Termination Time (seconds)
0.43693	0.005
0.62649	0.01
0.91743	0.1

Table 6.13: Interval-arithmetic hyperrectangle algorithm: characterization results for $f_1 = x^3 + y^2, f_2 = x^2 + y^3$ with $x \in [1,2], y \in [1,2]$

6.3 Conclusion

From the results of the experiments, for the polytope-based algorithm, it is observed that the chosen resolution introduces a trade-off. In all of the experiments, it is observed that increasing the resolution up to a certain threshold increases the percentage of characterized area when the termination time is kept constant. However, when the resolution is increased

more than this threshold, the percentage of characterized area decreases. It is also observed that there are factors that increase the average calculation time of affine approximations which has a negative impact on the scalability of the algorithm. These factors are observed to be increasing the termination time, increasing the number of operations in the performance functions and increasing the number of parameters in the performance functions. With increasing termination time, the number of parameter space divisions increases and this results in more complex grid points. Exact arithmetic operations with complex grid points result in an increased computation time which increases the average calculation time of affine approximations. With increasing number of operations in the performance functions, the number of exact arithmetic operations that is needed to calculate affine approximations increases which results in an increased average calculation time of affine approximations. With increasing number of parameters in the performance functions, the number of grid points in the mesh increases exponentially which results in an increased number of exact operations for the calculation of affine approximations.

For the monotone hyperrectangle-based algorithm, the factors that are listed for the polytope algorithm do not result in a significant change in the average calculation time of bounds. The main reason for this is that the number of exact arithmetic operations needed to calculate a bound is much more smaller than the number of exact arithmetic operations needed to calculate an affine approximation.

It is observed from the results of the experiments that the hyperrectangle-based algorithms always characterize more area per second than polytope algorithm and the monotone hyperrectangle algorithm characterizes slightly more area than the hyperrectangle algorithm that uses interval arithmetic. However, it should be noted that the monotone hyperrectangle algorithm is only limited to be used for performance functions that are monotone while interval arithmetic is not limited to such performance functions.

Chapter 7

LSAT Extension

In this chapter, we propose a parametric extension for the critical-path analysis performed by LSAT. In the first section, we provide a brief description on how LSAT performs critical-path analysis. In the second section, we provide the description of the proposed parametric extension.

7.1 Critical-Path Analysis of LSAT Models with Fixed Action Durations

In this section, we summarize the basics of the critical-path analysis performed by LSAT. Further details about this analysis and the modeling concepts used in LSAT can be found in [42].

Activities, as a directed acyclic graph, have a special structure. In an activity, each resource is claimed not more than once and each resource is released not more than once. This can be seen in Figure 3.6. As shown in Figure 3.4, this example activity uses three resources: Robot1, Robot2 and Robot3. Nodes n_1 and n_3 are the corresponding claim and release nodes for Robot1 while n_4, n_6 and n_7, n_9 are the corresponding nodes for Robot2 and Robot3, respectively.

In LSAT, system behavior is modeled using activity sequences. Figure 7.1 shows two activities A_1 and A_2. The values at the center of the nodes show the fixed durations of the corresponding actions. An example activity sequence, using these activities, is $A_1 \cdot A_2$. The execution of an activity sequence starts from the first activity, for our example A_1. Each node can only execute if the executions of all of its predecessor nodes are complete. An activity needs to claim a resource before it can execute action nodes that use that resource. For our example, A_2 needs to claim resource r_1 before it can execute its node with action (e, p_2). However, before A_2 claims r_1, r_1 needs to be released by A_1 considering that A_1 is executed before A_2. Therefore, in an activity, every claim node is succeeded by a release node on the corresponding resource. Because of the special structure of the activities, it is possible to represent an activity sequence by using a single concatenated activity. In a concatenation, two activities are connected by replacing the release and claim node pair of each resource by dependencies. In $A_1 \cdot A_2$, A_1 is followed by A_2. Therefore, the release nodes n_4, n_8 of A_1 and the claim nodes n_9, n_{12} of A_2 are removed

and the predecessor(s) of each claim node is connected with a dependency to the successor node(s) of the corresponding release node. This can be seen in Figure 7.2 as n_{19} is connected to n_{20} with a dependency and n_{24} is connected to both n_{25} and n_{26} with dependencies.

To find the critical path(s) of an activity sequence, such as $A_1 \cdot A_2$, an obvious way is to first find the concatenated activity. Then, we can use well-known algorithms that find the critical path(s) of directed acyclic graphs with fixed node durations. However, LSAT uses a more efficient approach to find the critical path(s) of activity sequences. Because of the special structure of activities, it is possible to represent the timing behavior of an activity using matrices. To find the timing matrix M_A of an activity $A = DAG(N_A, \rightarrow_A)$, we find the starting time $start(n)$ and ending time $end(n)$ of each node $n \in N_A$. The starting time of a node is equal to the maximum of the ending times of its predecessors and the ending time of a node is equal to its starting time plus its duration. An algorithm for computing these timing matrices automatically can be found in [18]. As an example, we calculate the timing matrix M_{A_1} of activity A_1. Let γ be a resource time stamp vector with each entry showing the availability time of the corresponding resource. Considering that A_1 and A_2 have two resources, for our example, $\gamma = [\gamma(r_1)\ \gamma(r_2)]^{\mathrm{T}}$.

$$
\begin{aligned}
end(n_1) &= \gamma(r_1) \\
end(n_5) &= \gamma(r_2) \\
end(n_2) &= \max(end(n_1)) + 1 = \gamma(r_1) + 1 \\
end(n_6) &= \max(end(n_5)) + 2 = \gamma(r_2) + 2 \\
end(n_3) &= \max(end(n_2), end(n_6)) + 3 \\
&= \max(\gamma(r_1) + 1, \gamma(r_2) + 2) + 3 \\
&= \max(\gamma(r_1) + 4, \gamma(r_2) + 5) \\
end(n_7) &= \max(end(n_6)) + 1 = \gamma(r_2) + 3 \\
end(n_4) &= end(n_3) = \max(\gamma(r_1) + 4, \gamma(r_2) + 5) \\
end(n_8) &= end(n_7) = \gamma(r_2) + 3
\end{aligned}
$$

The ending times of the release nodes can be written in the following normal form:

$$
\begin{aligned}
end(n_4) &= \max(\gamma(r_1) + 4, \gamma(r_2) + 5) \\
end(n_8) &= \max(\gamma(r_1) + -\infty, \gamma(r_2) + 3).
\end{aligned}
$$

These equations show that $end(n_4)$ is dependent on both the availability time of r_1 ($\gamma(r_1)$) and the availability time of r_2 ($\gamma(r_2)$). This can be seen in Figure 7.1. In A_1, n_4 can only be executed after the execution of n_3 and n_3 can only be executed after the execution of n_2 and n_6. Because the execution of n_2 is dependent on the availability time of r_1, $end(n_4)$ is also dependent on the availability time of r_1. A similar reasoning can be applied for n_6. The equations show that $end(n_8) = \max(\gamma(r_1) + -\infty, \gamma(r_2) + 3)$. The part with $\gamma(r_1) + -\infty$ shows that the execution of n_8 is not dependent on the availability time of r_1. The corresponding timing matrix M_{A_1} is the following:

$$
\boldsymbol{M}_{A_1} = \begin{bmatrix} 4 & 5 \\ -\infty & 3 \end{bmatrix}
$$

It should be noted that the entries of a timing matrix capture the longest/critical paths between resource claims and releases; that is, $\boldsymbol{M}_{i,j}$ is the duration of the longest path from the claim of resource r_j till the release of resource r_i. In A_1, there is one path from the claim of r_1 till the release of r_1 (n_1, n_2, n_3, n_4), so $[\boldsymbol{M}_{A_1}]_{1,1}$ is equal to 4 which is the duration of this path. The same reasoning applies for $[\boldsymbol{M}_{A_1}]_{1,2}$ and $[\boldsymbol{M}_{A_1}]_{2,2}$. $[\boldsymbol{M}_{A_1}]_{2,1}$ is equal to $-\infty$, because there is no path from the claim of resource r_1 to the release of resource r_2.

LSAT finds the critical path(s) of an activity sequence using the individual timing matrices of the activities in the sequence. First, LSAT finds the updated resource time stamp vectors after the execution of the activities in the sequence. These updated resource time stamp vectors are found by using the initial resource time stamp vector and the timing matrices of the activities in the sequence. For an initial resource time stamp vector γ, let γ' be the updated resource time stamp vector after the execution of an activity A. A resource r_i cannot be released before all the longest paths, from the claim of any resource r_j to the release of r_i, are executed. This means that the sum of $[\boldsymbol{M}_A]_{i,j}$ and $\gamma(r_j)$, for any resource r_j, limits the earliest time possible for the release of r_i. Therefore, the earliest time possible for the release of r_i $(\gamma'(r_i))$ is the maximum of these summations.

For our example, let $\gamma_0 = [0\ 0]$ be the initial resource time stamp. Then, the updated resource time stamp vector γ_1 after the execution of activity A_1 is the following:

$$\gamma_1(r_1) = \max([M_{A_1}]_{1,1} + \gamma_0(r_1), [M_{A_1}]_{1,2} + \gamma_0(r_2)) = \max(4 + 0, 5 + 0) = 5$$
$$\gamma_1(r_2) = \max([M_{A_1}]_{2,1} + \gamma_0(r_1), [M_{A_1}]_{2,2} + \gamma_0(r_2)) = \max(-\infty + 0, 3 + 0) = 3$$

A_2 is executed after the execution of A_1. Therefore, the final resource time stamp vector γ_2 after the execution of activity A_2 is the following:

$$\boldsymbol{M}_{A_2} = \begin{bmatrix} 2 & 3 \\ 2 & 3 \end{bmatrix}$$

$$\gamma_2(r_1) = \max([M_{A_2}]_{1,1} + \gamma_1(r_1), [M_{A_2}]_{1,2} + \gamma_1(r_2)) = \max(2 + 5, 3 + 3) = 7$$
$$\gamma_2(r_2) = \max([M_{A_2}]_{2,1} + \gamma_1(r_1), [M_{A_2}]_{2,2} + \gamma_1(r_2)) = \max(2 + 5, 3 + 3) = 7$$

Using the obtained resource time stamp vectors and the timing matrices of the activities in the sequence, LSAT finds the critical-path segments in each activity. This is done by tracing the matrix entries, in the obtained resource time stamp vectors and the timing matrices of the activities, that contribute to the duration of the critical path(s). The duration of the critical path(s) is the maximum of the resource availability times in the final resource time stamp vector.

In our example, the duration of the critical path(s) is equal to $\max(\gamma_2(r_1), \gamma_2(r_2)) = \max(7, 7) = 7$. Both of the availability times $\gamma_2(r_1)$ and $\gamma_2(r_2)$ are equal to the duration of the critical path(s). Therefore, all the entries, in γ_1 and $\boldsymbol{M}_{A_2}$, that contribute to $\gamma_2(r_1)$ or $\gamma_2(r_2)$ need to be found. These entries are $\gamma_1(r_1)$, $[M_{A_2}]_{1,1}$ and $[M_{A_2}]_{2,1}$. The entries, in γ_0 and $\boldsymbol{M}_{A_1}$, that contribute to $\gamma_1(r_1)$ are $\gamma_0(r_2)$ and $[M_{A_1}]_{1,2}$. Then, in A_1, the longest path(s) from the claim of r_2 to the release of r_1 (whose duration(s) is equal to $[M_{A_1}]_{1,2}$) is the critical-path segment(s). In A_2, the longest path(s) from the claim of r_1 to the release of r_1

(whose duration(s) is equal to $[M_{A_2}]_{1,1}$) and the longest path(s) from the claim of r_1 to the release of r_2 (whose duration(s) is equal to $[M_{A_2}]_{2,1}$) are the critical-path segments. This can be seen in Figures 7.1 and 7.2. In $A_1 \cdot A_2$, there are two critical paths $n_{22}, n_{23}, n_{19}, n_{20}, n_{21}$ and $n_{22}, n_{23}, n_{19}, n_{20}, n_{28}$. In A_1, the duration of the path n_5, n_6, n_3, n_4 is equal to $[M_{A_1}]_{1,2}$ and this path (without the release node n_4) is equal to a segment of the critical paths n_{22}, n_{23}, n_{19}. In A_2, the durations of the paths n_9, n_{10}, n_{11} and n_9, n_{10}, n_{16} are equal to $[M_{A_2}]_{1,1}$ and $[M_{A_2}]_{2,1}$, respectively, and these paths (without the claim node n_9) are equal to segments of the critical paths n_{20}, n_{21} and n_{20}, n_{28}, respectively.

To find the optimal activity sequence for a system, LSAT analyses the timing behavior of different activitiy sequences. By using the explained modular activity-based approach, rather than finding the concatenated activity of each activity sequence, LSAT re-uses the partial analyses across different activity sequences which results in higher efficiency.

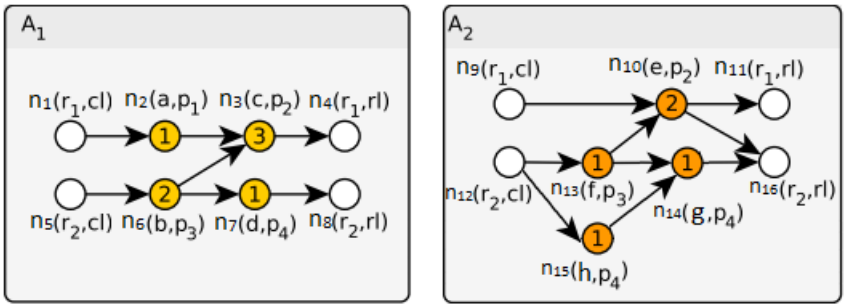

Figure 7.1: Activities A_1 and A_2

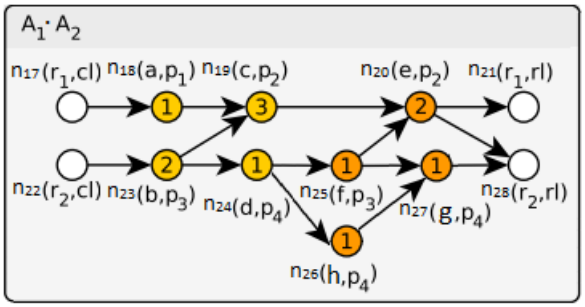

Figure 7.2: Activity $A_1 \cdot A_2$

7.2 Critical-Path Analysis of Parameterised LSAT Models

In the previous section, we provide a brief description on how LSAT performs the critical-path analysis of models with fixed action durations. In this section, we provide an approach for the parametric critical-path analysis of parameterised LSAT models.

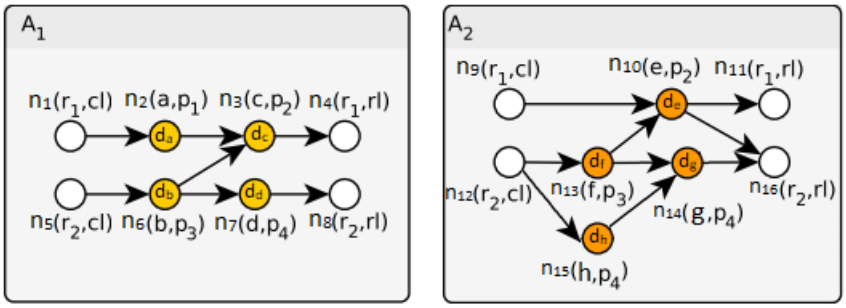

Figure 7.3: Parameterised activities A_1 and A_2

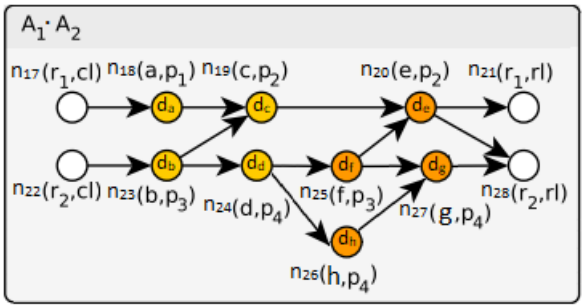

Figure 7.4: Parameterised activity $A_1 \cdot A_2$

As stated in the previous section, to find the critical path(s) of an activity sequence, LSAT uses the individual timing matrices of the activities in the sequence. We propose a similar approach for the parametric critical-path analysis.

Figure 7.3 shows the parameterised activities A_1 and A_2 and Figure 7.4 shows the parameterised concatenated activity $A_1 \cdot A_2$. The functions in the center of the nodes represent the duration functions of the corresponding actions. Then, the following defines the duration functions of the nodes:

$$
\begin{aligned}
d_{n_2}(x,y) = d_{n_{18}}(x,y) &= d_{\mathrm{a}}(x,y) = x^2 + y^2 + 3y \\
d_{n_6}(x,y) = d_{n_{23}}(x,y) &= d_{\mathrm{b}}(x,y) = y^3 + x \\
d_{n_3}(x,y) = d_{n_{19}}(x,y) &= d_{\mathrm{c}}(x,y) = y^4 + 4x \\
d_{n_7}(x,y) = d_{n_{24}}(x,y) &= d_{\mathrm{d}}(x,y) = 3x + 1 \\
d_{n_{10}}(x,y) = d_{n_{20}}(x,y) &= d_{\mathrm{e}}(x,y) = x^2 - x + 3 \\
d_{n_{13}}(x,y) = d_{n_{25}}(x,y) &= d_{\mathrm{f}}(x,y) = x^2 + 3 \\
d_{n_{14}}(x,y) = d_{n_{27}}(x,y) &= d_{\mathrm{g}}(x,y) = y^2 + 5y \\
d_{n_{15}}(x,y) = d_{n_{26}}(x,y) &= d_{\mathrm{h}}(x,y) = y^3 + 2
\end{aligned}
$$

In an activity with fixed action durations, the starting time of a node n with predecessors n_{p1} and n_{p2} is calculated by finding the maximum of the ending times of n_{p1} and

n_{p2}. The ending times of n_{p1} and n_{p2} are of the form $\max(\gamma(r_1) + c_{11}, \gamma(r_2) + c_{12}, ...)$ and $\max(\gamma(r_1) + c_{21}, \gamma(r_2) + c_{22}, ...)$, respectively. Therefore, the starting time of n is $\max(\gamma(r_1) + \max(c_{11}, c_{21}), \gamma(r_2) + \max(c_{12}, c_{22}), ...)$. The maximum of these fixed values, such as $\max(c_{11}, c_{21})$, is also a fixed value.

For parameterised activities, similar to the activities with fixed action durations, we represent the timing behavior by using parameterised timing matrices. However, considering that the durations are functions rather than fixed values, the duration functions of paths can be maximum at different points in the domain of interest and we need to keep track of every path that is potentially a part of a critical path at any point in the domain of interest. Therefore, we use *starting duration-function set* $start(n)$ and *ending duration-function set* $end(n)$ to represent the starting time and ending time of a node n in a parameterised activity. In order to eliminate some of the paths that are not a part of a critical path at any point in the domain of interest, we find $start(n)$ by comparing the duration functions in the ending duration-function sets of the predecessors of n. This comparison is done by finding the lower bounds of the differences between these duration functions.

Given parameterised activity $A = DAG(N_A, \rightarrow_A)$, hyperrectangular parameter space of interest $\mathbb{D}$ and parameterised resource time stamp vector γ, let $Pred(n)$ be the set of all predecessors of a node $n \in N_A$ and $end_{Pred}(n)$ be the set of all duration functions in the ending duration-function sets of the predecessors of n which means $end_{Pred}(n) = \bigcup_{n_{in} \in Pred(n)} end(n_{in})$. Additionally, let $lb_{f_j - f_i}$ be a lower bound of the difference of the duration functions f_i and f_j that satisfies $f_j(p) - f_i(p) \geq lb_{f_j - f_i}\ \forall p \in \mathbb{D}$. Then, the starting duration-function set $start(n)$ and the ending duration-function set $end(n)$ is found as follows:

$$start(n) = \begin{cases} \gamma(r) & \text{if } n \text{ has a cl/rl action} \\ \{f_i \mid f_i \in end_{Pred}(n) \wedge lb_{f_j - f_i} \leq 0\ \forall f_j \in end_{Pred}(n)\} & \text{otherwise} \end{cases}$$

$$end(n) = \{f + d(n) \mid f \in start(n)\}$$

In the above definition, each duration function $f \in end_{Pred}(n)$ is the duration function of a different path with a predecessor of n as the last node. If, for a duration function $f_i \in end_{Pred}(n)$, $lb_{f_j - f_i} > 0$ is true for any $f_j \in end_{Pred}(n)$, then $f_j(p) - f_i(p) \geq lb_{f_j - f_i} > 0\ \forall p \in \mathbb{D}$ is satisfied. This proves that the path with the duration function f_i is not a part of a critical path at any point $p \in \mathbb{D}$. Therefore, in $start(n)$, we only include duration functions $f_i \in end_{Pred}(n)$ that satisfy $lb_{f_j - f_i} \leq 0\ \forall f_j \in end_{Pred}(n)$.

As discussed in Chapter 5, the lower bounds can be calculated by using interval arithmetic. If the monotonicity requirements are satisfied, then it is also possible to use the corners of $\mathbb{D}$ to find the exact maximum and minimum, just like we presented in Chapter 5. However, in practical models, such monotonicity requirements are not always satisfied.

Now, as an example, we calculate $start(n)$ and $end(n)$ of all nodes $n \in N_{A_2}$ and the corresponding parameterised timing matrix M_{A_2} for parameter space $\mathbb{D} = \mathbb{D}_x \times \mathbb{D}_y$ where $\mathbb{D}_x = [4, 6]$ and $\mathbb{D}_y = [1, 2]$. We use interval arithmetic to find the bounds of the differences between the duration functions.

The nodes n_9 and n_{12} have claim (cl) actions.

$$\begin{aligned} start(n_9) &= \gamma(r_1) \\ end(n_9) &= \{f + 0 \mid f \in \gamma(r_1)\} = \gamma(r_1) \\ start(n_{12}) &= \gamma(r_2) \\ end(n_{12}) &= \{f + 0 \mid f \in \gamma(r_2)\} = \gamma(r_2) \end{aligned}$$

The nodes n_{13} and n_{15} have only one predecessor node. Therefore, no comparison is needed.

$$\begin{aligned} start(n_{13}) &= end(n_{12}) \\ end(n_{13}) &= \{f + d_{n_{13}} \mid f \in end(n_{12})\} = \{f + d_{n_{13}} \mid f \in \gamma(r_2)\} \\ start(n_{15}) &= end(n_{12}) \\ end(n_{15}) &= \{f + d_{n_{15}} \mid f \in end(n_{12})\} = \{f + d_{n_{15}} \mid f \in \gamma(r_2)\} \end{aligned}$$

By checking the ending duration-function sets of the predecessors of n_{10}, $end(n_9) = \gamma(r_1)$ and $end(n_{13}) = \{f + d_{n_{13}} \mid f \in \gamma(r_2)\}$, we see that $end(n_9)$ only depends on $\gamma(r_1)$, while $end(n_{13})$ only depends on $\gamma(r_2)$. Therefore, it is not possible to compare the duration functions $f_i \in end(n_9)$ with the duration functions $f_j \in end(n_{13})$ which means that no elimination is possible for $start(n_{10})$.

$$\begin{aligned} start(n_{10}) &= end(n_9) \cup end(n_{13}) = \{f \mid f \in \gamma(r_1)\} \cup \{f + d_{n_{13}} \mid f \in \gamma(r_2)\} \\ end(n_{10}) &= \{f + d_{n_{10}} \mid f \in start(n_{10})\} = \{f + d_{n_{10}} \mid f \in \gamma(r_1)\} \cup \{f + d_{n_{13}} + d_{n_{10}} \mid f \in \gamma(r_2)\} \\ start(n_{11}) &= end(n_{10}) \\ end(n_{11}) &= start(n_{11}) = \{f + d_{n_{10}} \mid f \in \gamma(r_1)\} \cup \{f + d_{n_{13}} + d_{n_{10}} \mid f \in \gamma(r_2)\} \end{aligned}$$

By checking the ending duration-function sets of the predecessors of n_{14}, $end(n_{13}) = \{f + d_{n_{13}} \mid f \in \gamma(r_2)\}$ and $end(n_{15}) = \{f + d_{n_{15}} \mid f \in \gamma(r_2)\}$, we see that both $end(n_{13})$ and $end(n_{15})$ depend on $\gamma(r_2)$. Therefore, we can make a comparison and check if we can do any elimination for $start(n_{14})$.

$$\begin{aligned} 9 \leq d_{n_{13}}(p) - d_{n_{15}}(p) \leq 36 \quad \forall p \in \mathbb{D} \\ lb_{d_{n_{13}} - d_{n_{15}}} = 9 > 0 \end{aligned}$$

Because of $lb_{d_{n_{13}} - d_{n_{15}}} = 9 > 0$, we know that there is a duration function $f_{big} \in \{f + d_{n_{13}} \mid f \in \gamma(r_2)\}$ for each duration function $f \in \{f + d_{n_{15}} \mid f \in \gamma(r_2)\}$ such that $f_{big}(p) > f(p)$ for all $p \in \mathbb{D}$. This proves that the path (n_{12}, n_{15}, n_{14}) is not a part of a critical path at any point in $\mathbb{D}$, because the value of the duration function of the path (n_{12}, n_{13}, n_{14}) is greater than the value of the duration function of (n_{12}, n_{15}, n_{14}) at any point in $\mathbb{D}$. Therefore, the duration functions in $end(n_{15})$ are eliminated.

$$\begin{aligned} start(n_{14}) &= end(n_{13}) = \{f + d_{n_{13}} \mid f \in \gamma(r_2)\} \\ end(n_{14}) &= \{f + d_{n_{13}} + d_{n_{14}} \mid f \in \gamma(r_2)\} \end{aligned}$$

For $start(n_{16})$, the ending duration-function sets of the predecessors are $end(n_{10}) = \{f + d_{n_{10}} \mid f \in \gamma(r_1)\} \cup \{f + d_{n_{13}} + d_{n_{10}} \mid f \in \gamma(r_2)\}$ and $end(n_{14}) = \{f + d_{n_{13}} + d_{n_{14}} \mid f \in \gamma(r_2)\}$.

We do the following comparisons.

$$-1 \leq d_{n13} + d_{n10} - d_{n13} - d_{n14} \leq 29$$
$$lb_{d_{n10}-d_{n14}} = -1 \leq 0$$
$$-29 \leq d_{n13} + d_{n14} - d_{n13} - d_{n10} \leq 1$$
$$lb_{d_{n14}-d_{n10}} = -29 \leq 0$$

Because of $lb_{d_{n10}-d_{n14}} = -1 \leq 0$ and $lb_{d_{n14}-d_{n10}} = -29 \leq 0$, no elimination can be done for $start(n_{16})$.

$$start(n_{16}) = end(n_{10}) \cup end(n_{14})$$
$$end(n_{16}) = start(n_{16}) =$$
$$\{f + d_{n10} \mid f \in \gamma(r_1)\} \cup \{f + d_{n13} + d_{n10} \mid f \in \gamma(r_2)\} \cup \{f + d_{n13} + d_{n14} \mid f \in \gamma(r_2)\}$$

The release nodes of A_2 are n_{11} and n_{16}. Because of $end(n_{11}) = \{f + d_{n10} \mid f \in \gamma(r_1)\} \cup \{f + d_{n13} + d_{n10} \mid f \in \gamma(r_2)\}$, $[M_{A_2}]_{11}$ is equal to $\{(d_{n10})\}$ and $[M_{A_2}]_{12}$ is equal to $\{(d_{n13} + d_{n10})\}$. Similarly, because of $end(n_{16}) = \{f + d_{n10} \mid f \in \gamma(r_1)\} \cup \{f + d_{n13} + d_{n10} \mid f \in \gamma(r_2)\} \cup \{f + d_{n13} + d_{n14} \mid f \in \gamma(r_2)\}$, $[M_{A_2}]_{21}$ is equal to $\{(d_{n10})\}$ and $[M_{A_2}]_{22}$ is equal to $\{(d_{n13} + d_{n10}), (d_{n13} + d_{n14})\}$. Then, the parameterised timing matrix of activity A_2 is the following:

$$M_{A_2} = \begin{bmatrix} \{(d_{n10})\} & \{(d_{n13} + d_{n10})\} \\ \{(d_{n10})\} & \{(d_{n13} + d_{n10}), (d_{n13} + d_{n14})\} \end{bmatrix}$$

To solve the parametric critical-path problem, defined in Section 3.3, for a parameterised activity sequence such as $A_1 \cdot A_2$, one way is to first find the corresponding concatenated activity $A = A_1 \cdot A_2$ and its parameterised timing matrix M_A. Then, the duration functions that are members of the elements $[M_A]_{i,j}$ are the duration functions of the maximal paths of A. It should be noted that we eliminate some of the maximal paths, that are not critical at all $p \in \mathbb{D}$, while constructing the matrix with duration function comparisons.

The parameterised timing matrix of the concatenated activity $A_1 \cdot A_2$ is given below.

$$M_{A_1 \cdot A_2} = \begin{bmatrix} fs_{1,1} & fs_{1,2} \\ fs_{2,1} & fs_{2,2} \end{bmatrix}$$

$$fs_{1,1} = \{(d_{n18} + d_{n19} + d_{n20})\}$$
$$fs_{1,2} = \{(d_{n23} + d_{n19} + d_{n20}), (d_{n23} + d_{n24} + d_{n25} + d_{n20})\}$$
$$fs_{2,1} = \{(d_{n18} + d_{n19} + d_{n20})\}$$
$$fs_{2,2} = \{(d_{n23} + d_{n19} + d_{n20}), (d_{n23} + d_{n24} + d_{n25} + d_{n20}), (d_{n23} + d_{n24} + d_{n25} + d_{n27})\}$$

Then, $\mathcal{F}_{maxPaths} = fs_{1,1} \cup fs_{1,2} \cup fs_{2,1} \cup fs_{2,2} = \{(d_{n18} + d_{n19} + d_{n20}), (d_{n23} + d_{n19} + d_{n20}), (d_{n23} + d_{n24} + d_{n25} + d_{n20}), (d_{n23} + d_{n24} + d_{n25} + d_{n27})\}$ is the set of the duration functions of the maximal paths of $A_1 \cdot A_2$. The duration functions of maximal paths $p_1 = (n_{17}, n_{18}, n_{19}, n_{20}, n_{21})$ and $p_2 = (n_{17}, n_{18}, n_{19}, n_{20}, n_{28})$ are $f_{p_1} = f_{p_2} = (d_{n18} + d_{n19} + d_{n20})$, the duration functions of maximal paths $p_3 = (n_{22}, n_{23}, n_{19}, n_{20}, n_{21})$ and $p_4 = (n_{22}, n_{23}, n_{19}, n_{20}, n_{28})$ are

$f_{p_3} = f_{p_4} = (d_{n_{23}} + d_{n_{19}} + d_{n_{20}})$ and etc. It should be noted that the duration function of maximal path $p_5 = (n_{22}, n_{23}, n_{24}, n_{26}, n_{27}, n_{28})$ is $f_{p_5} = (d_{n_{23}} + d_{n_{24}} + d_{n_{26}} + d_{n_{27}})$ and it is not a member of $\mathcal{F}_{maxPaths}$. Because we concluded that it is not critical at all points $p \in \mathbb{D}$, as a result of the comparisons we did while we constructed matrix $M_{A_1 \cdot A_2}$.

Now, we have the duration functions of the maximal paths of $A_1 \cdot A_2$. Therefore, we can solve the parametric critical-path problem for $A_1 \cdot A_2$ by solving the following characterization problem: find the mapping $C : \mathbb{P} \to \wp(\mathcal{F}) \setminus \{\emptyset\}$ where $\mathbb{P} = \mathbb{D}$ and $\mathcal{F} = \mathcal{F}_{maxPaths}$.

There is a more efficient approach to solve the parametric critical-path problem for parameterised activity sequences. Similar to the case with fixed action durations, we use the individual parameterised timing matrices of the activities in the activity sequence. By using these individual matrices, we find the duration functions of the maximal paths of the concatenated activity without calculating the concatenated activity.

Given a parameterised resource time stamp vector γ, let each entry of the parameterised resource time stamp vector γ' show the availability of the corresponding resource after the execution of activity A. We can calculate γ' by using parameterised matrix multiplication, $\gamma' = M_A \otimes \gamma$, which is given as follows.

Given hyperrectangular domain $\mathbb{D}$, $m \times p$ parameterised timing matrix M_A and $p \times 1$ parameterised resource time stamp vector γ, the elements of the resulting parameterised resource time stamp vector $\gamma' = M_A \otimes \gamma$ are determined by:

$$\gamma'(r_i) = \{f_1 \mid f_1 \in \mathcal{F}_{Com} \wedge lb_{f_2 - f_1} \leq 0 \; \forall f_2 \in \mathcal{F}_{Com} \wedge \mathcal{F}_{Com} = \bigcup_{k=1}^{p} \{f_A + f_B \mid f_A \in [M_A]_{ik} \wedge f_B \in \gamma(r_k)\}\}$$

The definition of the parameterised matrix multiplication is quite similar to the definition of the starting duration-function set. However, there are significant differences. In the calculation of the starting duration-function set, we only use the dependencies on the availability of the resources. Because we are interested in the individual timing behavior of an activity. In the parameterised matrix multiplication, the timing behavior (parameterised timing matrix) of an activity is already known and, by using a concrete parameterised resource time stamp vector with duration functions in its entries, we find the availability of the resources after the execution of the activity. Each duration function $f \in \mathcal{F}_{Com}$ is the summation of the duration function (f_A) of a path that starts from the claim of a resource r_k and ends with the release of r_i, and a duration function (f_B) whose value is potentially equal to the availability time of r_k at some point in $\mathbb{D}$. By comparing these duration functions, we eliminate any duration function $f_A + f_B$ whose value is never equal to the availability time of r_i after the execution of activity A.

For an activity sequence, if we continue finding the parameterised resource time stamp vectors following the sequential execution of the activities then, at the end, we find the duration functions of the maximal paths of the concatenated activity. Now, as an example, we show how we can calculate the duration functions of the maximal paths of $A_1 \cdot A_2$ using the matrices M_{A_1}

and M_{A_2}. We assume an initial parameterised resource time stamp vector $\gamma_0 = [\{0\}\ \{0\}]$.

$$M_{A_1} = \begin{bmatrix} \{(d_{n_2} + d_{n_3})\} & \{(d_{n_6} + d_{n_3})\} \\ -\infty & \{(d_{n_2} + d_{n_3})\} \end{bmatrix}$$

$$M_{A_2} = \begin{bmatrix} \{(d_{n_{10}})\} & \{(d_{n_{13}} + d_{n_{10}})\} \\ \{(d_{n_{10}})\} & \{(d_{n_{13}} + d_{n_{10}}), (d_{n_{13}} + d_{n_{14}})\} \end{bmatrix}$$

$$\gamma_1 = M_{A_1} \otimes \gamma_0 = \begin{bmatrix} \{(d_{n_2} + d_{n_3})\} & \{(d_{n_6} + d_{n_3})\} \\ -\infty & \{(d_{n_6} + d_{n_7})\} \end{bmatrix} \otimes \begin{bmatrix} \{0\} \\ \{0\} \end{bmatrix} = \begin{bmatrix} \{(d_{n_2} + d_{n_3})\} \\ \{(d_{n_6} + d_{n_7})\} \end{bmatrix}$$

To find $\gamma_1(r_1)$, the duration functions $(d_{n_2} + d_{n_3} + 0)$ and $(d_{n_6} + d_{n_3} + 0)$ are compared. Using interval arithmetic, we find the bound $6 \leq d_{n_2} - d_{n_6} \leq 41$. Because of $lb_{d_{n_2} - d_{n_6}} = 6 > 0$, it is proven that the value of the duration function of the path (n_5, n_6, n_3, n_4) is greater than the value of the duration function of the path (n_1, n_2, n_3, n_4) at any point in $\mathbb{D}$ for the initial parameterised resource time stamp vector $\gamma_0 = [\{0\}\ \{0\}]$. Therefore, (n_5, n_6, n_3, n_4) is not a part of a critical path at any point in $\mathbb{D}$ and $(d_{n_6} + d_{n_3} + 0)$ is eliminated.

$$\gamma_2 = M_{A_2} \otimes \gamma_1 = \begin{bmatrix} \{(d_{n_{10}})\} & \{(d_{n_{13}} + d_{n_{10}})\} \\ \{(d_{n_{10}})\} & \{(d_{n_{13}} + d_{n_{10}}), (d_{n_{13}} + d_{n_{14}})\} \end{bmatrix} \otimes \begin{bmatrix} \{(d_{n_2} + d_{n_3})\} \\ \{(d_{n_6} + d_{n_7})\} \end{bmatrix} = \begin{bmatrix} \gamma_{fs}(r_1) \\ \gamma_{fs}(r_2) \end{bmatrix}$$

where $\gamma_{fs}(r_1) = \{(d_{n_2} + d_{n_3} + d_{n_{10}}), (d_{n_6} + d_{n_7} + d_{n_{13}} + d_{n_{10}})\}$ and $\gamma_f(r_2) = \{(d_{n_2} + d_{n_3} + d_{n_{10}}), (d_{n_6} + d_{n_7} + d_{n_{13}} + d_{n_{10}}), (d_{n_6} + d_{n_7} + d_{n_{13}} + d_{n_{14}}\}$.

We can see that $\gamma_{fs}(r_1) = fs_{1,1} \cup fs_{1,2}$ and $\gamma_f(r_2) = fs_{2,1} \cup fs_{2,2}$. Therefore, we obtain the same set of duration functions with $M_{A_1 \cdot A_2}$.

To find the duration functions of the maximal paths of an activity sequence, using individual parameterised timing matrices and parameterised matrix multiplication is more efficient than directly calculating the concatenated matrix and the corresponding parameterised timing matrix. Because we need to calculate the individual parameterised timing matrices only once, and then we can re-use these parameterised timing matrices to find the parameterised timing matrix of any activity sequence.

7.3 Conclusion

In this chapter, the first section provides a brief summary about how LSAT performs critical-path analysis when the durations of nodes are fixed values. By using the special structure of the activities (obtained by soundness rules), the timing behavior of an activity is represented using a timing matrix. To find the critical path of an activity sequence, LSAT uses the timing matrices of the individual activities in the sequence. With this modular approach, it is possible to use the partial analysis of each activity in the critical-path analysis of different activity sequences.

In the second section, we proposed a parametric extension to this modular approach. With parameterised durations of nodes, a different path can be the path with the maximum duration value at different points in the parameter space. Therefore, in parameterised

activity sequences, we keep track of every path that has the potential to be the path with the maximum duration value at some point in the parameter space. Then, using a similar modular approach, we find all the potentially critical paths of an activity sequence by using the parameterised timing matrices of the individual activities in the sequence. After these potentially critical paths of the activity sequence are found with their corresponding duration functions. The parametric critical-path problem for this activity sequence can be solved by solving the corresponding characterization problem (with performance functions equal to the duration functions of the potentially critical paths) using the presented characterization algorithms.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

This thesis focuses on the parametric critical-path analysis of parameterized LSAT models. Two main contributions are the following:

- We designed and implemented a polytope-based characterization algorithm and a hyperrectangle-based characterization algorithm targeted towards the third-order point-to-point motion profiles that are used in LSAT. We also designed and implemented a more generic hyperrectangle-based characterization algorithm that uses interval arithmetic.
- We proposed a parametric extension to the critical-path analysis performed by LSAT.

The algorithms that are targeted towards the third-order point-to-point motion profiles were designed according to the properties of the third-order point-to-point motion profiles. We analysed the scalability and performances of these algorithms using generic performance functions. Because of its exponential time complexity, the polytope algorithm is targeted towards performance functions with few parameters. The experiments showed that, even for performance functions with few parameters, the monotonic hyperrectangle algorithm performs better than the polytope algorithm. Apart from the exponential complexity, the rational numbers that are used for exact arithmetic caused significant increases in the calculation time of affine approximations that are used in the polytope algorithm. The experiments also showed that the interval-arithmetic hyperrectangle algorithm performs better than the polytope algorithm and slightly worse than the monotonic hyperrectangle algorithm. We concluded that, considering the general applicability of interval arithmetic, this slight decrease in performance can be acceptable.

We provided a brief summary of the critical-path analysis performed by LSAT for non-parameterised models. LSAT uses the special structure of the activities to find the critical paths of activity sequences without calculating the concatenated activity. Using the individual timing matrices of the activities in the sequence, LSAT finds the critical path segments in different activities independent of each other. This modular activity-based approach provides efficiency since the partial analyses are re-used to analyse the timing behavior of different activity sequences. We proposed a parametric extension to this analysis.

For the parameterised LSAT models, we proposed to use parametric timing matrices that are calculated by using the bounds of the duration functions of the actions in the activities. By using a parameterised matrix multiplication, we calculated the parametric timing matrix of the concatenated activity without calculating the concatenated activity. Then, the parametric critical-path problem for this activity sequence can be solved by solving the corresponding characterization problem, with the set of performance functions being equal to the set of duration functions in the parametric timing matrix of the concatenated activity, using the three developed characterization algorithms.

8.2 Future Work

For the three developed characterization algorithms, the number of experiments can be increased by using different types of performance functions for a more thorough analysis of the performance and scalability of the algorithms. A significant improvement in the polytope algorithm can be achieved by eliminating the increase in the computation time of affine approximations caused by the rational numbers. This may be achieved by using floating-point arithmetic and calculating the corresponding error, so that we can still give correct results without using computationally expensive rational numbers.

The proposed parametric extension can be implemented and its performance can be analysed by experiments. Considering the results of the experiments in this thesis, the hyperrectangle-based algorithm using interval arithmetic is the best candidate for this parametric extension. By analysing the characteristics of the duration functions of the actions that are used in parameterised LSAT models, we can find contractors from the literature to further increase the performance of this algorithm.

Bibliography

[1] V. Alimguzhin, F. Mari, I. Melatti, I. Salvo, and E. Tronci. Linearizing Discrete-Time Hybrid Systems. *IEEE Transactions on Automatic Control*, 62(10):5357–5364, 2017. 6

[2] A. Bemporad and C. Filippi. An Algorithm for Approximate Multiparametric Convex Programming. *Computational optimization and applications*, 35(1):87–108, 2006. 5

[3] F. Benhamou, F. Goualard, L. Granvilliers, and J. Puget. Revising hull and box consistency. In *ICLP*, volume 99, pages 230–244, 1999. 5

[4] F. Benhamou, F. Goualard, É. Languénou, and M. Christie. Interval Constraint Solving for Camera Control and Motion Planning. *ACM Transactions on Computational Logic (TOCL)*, 5(4):732–767, 2004. 5

[5] F. Benhamou and L. Granvilliers. Continuous and Interval Constraints. *Foundations of Artificial Intelligence*, 2:571–603, 2006. 5

[6] B. Büeler, A. Enge, and K. Fukuda. Exact volume computation for polytopes: a practical study. In *Polytopes—combinatorics and computation*, pages 131–154. Springer, 2000. 40

[7] G. Chabert and L. Jaulin. Contractor programming. *Artificial Intelligence*, 173(11):1079–1100, 2009. 5

[8] V. M. Charitopoulos. Parametric Optimisation: 65 years of developments and status quo. In *Uncertainty-aware Integration of Control with Process Operations and Multi-parametric Programming Under Global Uncertainty*, pages 9–45. Springer, 2020. 5

[9] V. M. Charitopoulos, L. G. Papageorgiou, and V. Dua. Nonlinear Model-Based Process Operation under Uncertainty Using Exact Parametric Programming. *Engineering*, 3(2):202–213, 2017. 5

[10] L. F. Domínguez and E. N. Pistikopoulos. A Novel mp-NLP Algorithm for Explicit/Multi-parametric NMPC. *IFAC Proceedings Volumes*, 43(14):539–544, 2010. 5

[11] V. Dua, K. P. Papalexandri, and E. N. Pistikopoulos. Global Optimization Issues in Multiparametric Continuous and Mixed-Integer Optimization Problems. *Journal of Global Optimization*, 30(1):59–89, 2004. 5

[12] I. A. Fotiou. *Parametric Optimization and Constrained Optimal Control for Polynomial Dynamical Systems*. PhD thesis, ETH Zurich, 2008. 5

[13] I. A. Fotiou, P. A. Parrilo, and M. Morari. Nonlinear parametric optimization using cylindrical algebraic decomposition. In *44th IEEE Conference on Decision and Control*, pages 3735–3740. IEEE, 2005. 5

[14] I. A. Fotiou, P. Rostalski, P. A. Parrilo, and M. Morari. Parametric optimization and optimal control using algebraic geometry methods. *International Journal of Control*, 79(11):1340–1358, 2006. 5

[15] K. Fukuda. Cddlib reference manual. *Report version 094i, McGill University, Montréal, Quebec, Canada*, 2003. 39

[16] K. Fukuda and A. Prodon. Double description method revisited. In *Franco-Japanese and Franco-Chinese Conference on Combinatorics and Computer Science*, pages 91–111. Springer, 1995. 39

[17] E. Garajová and M. Meciar. Solving and Visualizing Nonlinear Set Inversion Problems. *Reliable Computing*, 22:105, 2016. 5

[18] M. Geilen. Synchronous dataflow scenarios. *ACM Transactions on Embedded Computing Systems (TECS)*, 10(2):1–31, 2011. 53

[19] A. H. Ghamarian, M. C. W. Geilen, T. Basten, and S. Stuijk. Parametric Throughput Analysis of Synchronous Data Flow Graphs. In *2008 Design, Automation and Test in Europe*, pages 116–121. IEEE, 2008. 4

[20] T. Granlund. The GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 6(0):10, 2016. 39

[21] J. F. Groote, A. Mathijssen, M. Reniers, Y. Usenko, and M. Van Weerdenburg. The formal specification language mcrl2. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2007. 2

[22] E. T. Hale. *Numerical Methods for d-Parametric Nonlinear Programming with Chemical Process Control and Optimization Applications*. The University of Texas at Austin, 2005. 5

[23] E. T. Hale and S. J. Qin. Multi-Parametric Nonlinear Programming and the Evaluation of Implicit Optimization Model Adequacy. *IFAC Proceedings Volumes*, 37(9):449–454, 2004. 5

[24] K. R. Heloue, S. Onaissi, and F. N. Najm. Efficient Block-Based Parameterized Timing Analysis Covering All Potentially Critical Paths. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(4):472–484, 2012. 4

[25] P. Herrero, P. Georgiou, C. Toumazou, B. Delaunay, and L. Jaulin. An Efficient Implementation of the SIVIA Algorithm in a High-Level Numerical Programming Language. *Reliable Computing*, 16:239–251, 2012. 5

[26] Timothy Hickey, Qun Ju, and Maarten H Van Emden. Interval arithmetic: From principles to implementation. *Journal of the ACM (JACM)*, 48(5):1038–1068, 2001. 32

[27] T. Hune, J. Romijn, M. Stoelinga, and F. Vaandrager. Linear parametric model checking of timed automata. *The Journal of Logic and Algebraic Programming*, 52:183–220, 2002. 4

[28] L. Jaulin. Interval constraint propagation with application to bounded-error estimation. *Automatica*, 36(10):1547–1552, 2000. 5

[29] L. Jaulin, M. Kieffer, I. Braems, and E. Walter. Guaranteed non-linear estimation using constraint propagation on sets. *International Journal of Control*, 74(18):1772–1782, 2001. 5

[30] L. Jaulin, M. Kieffer, O. Didrit, and É. Walter. *Applied Interval Analysis*. Springer, 2001. 5

[31] L. Jaulin and E. Walter. Set Inversion via Interval Analysis for Nonlinear Bounded-error Estimation. *Automatica*, 29(4):1053–1064, 1993. 4

[32] Z. Jin, Q. Shen, and S. Z. Yong. Mesh-based piecewise affine abstraction with polytopic partitions for nonlinear systems. *IEEE Control Systems Letters*, 5(5):1543–1548, 2020. 6, 15, 16, 17, 23

[33] T. A. Johansen. Approximate explicit receding horizon control of constrained nonlinear systems. *Automatica*, 40(2):293–300, 2004. 5

[34] Y. Koren. General RMS Characteristics. Comparison with Dedicated and Flexible Systems. In *Reconfigurable Manufacturing Systems and Transformable Factories*, pages 27–45. Springer, 2006. 1

[35] P. Lambrechts, M. Boerlage, and M. Steinbuch. Trajectory planning and feedforward design for electromechanical motion systems. *Control Engineering Practice*, 13(2):145–157, 2005. 4

[36] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *International journal on software tools for technology transfer*, 1(1):134–152, 1997. 2

[37] J. Leverenz, M. Xu, and M. M. Wiecek. Multiparametric optimization for multidisciplinary engineering design. *Structural and Multidisciplinary Optimization*, 54(4):795–810, 2016. 5

[38] K. Meixner, R. Rabiser, and S. Biffl. Towards Modeling Variability of Products, Processes and Resources in Cyber-Physical Production Systems Engineering. In *23rd International Systems and Software Product Line Conference-Volume B*, pages 49–56. ACM, 2019. 1

[39] P. S. V. Nataraj and M. M. Deshpande. An Improved Algorithm for Set Inversion using Interval Analysis with Application to Control System. In *2006 IEEE International Conference on Industrial Technology*, pages 1548–1552. IEEE, 2006. 5

[40] S. Rebennack and J. Kallrath. Continuous Piecewise Linear Delta-Approximations for Bivariate and Multivariate Functions. *Journal of Optimization Theory and Applications*, 167(1):102–117, 2015. 6

[41] F. Rossi, P. Van Beek, and T. Walsh. *Handbook of constraint programming*. Elsevier, 2006. 31

[42] B. van der Sanden. *Performance analysis and optimization of supervisory controllers*. PhD thesis, Eindhoven University of Technology, 2018. 52

[43] B. van der Sanden, J. Bastos, J. Voeten, M. Geilen, M. Reniers, T. Basten, J. Jacobs, and R. Schiffelers. Compositional Specification of Functionality and Timing of Manufacturing Systems. In *2016 Forum on Specification and Design Languages (FDL)*, pages 1–8. IEEE, 2016. 7

[44] B. van der Sanden, Y. Blankenstein, R. Schiffelers, and J. Voeten. LSAT: Specification and Analysis of Product Logistics in Flexible Manufacturing Systems. In *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pages 1–8. IEEE, 2021. 2, 7

[45] B. van der Sanden, Y. Li, J. van den Aker, B. Akesson, T. Bijlsma, M. Hendriks, K. Triantafyllidis, J. Verriet, J. Voeten, and T. Basten. Model-Driven System-Performance Engineering for Cyber-Physical Systems: Industry Session Paper. In *2021 International Conference on Embedded Software (EMSOFT)*, pages 11–22. IEEE, 2021. 1

[46] J. van Pinxten, M. Geilen, M. Hendriks, and T. Basten. Parametric Critical Path Analysis for Event Networks With Minimal and Maximal Time Lags. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2697–2708, 2018. 4, 5, 6

[47] J. P. M. Voeten. *POOSL: An object-oriented specification language for the analysis and design of hardware/software systems*. Eindhoven University of Technology, Faculty of Electrical Engineering, 1995. 2

[48] E. Walter and M. Kieffer. Interval Analysis for Guaranteed Nonlinear Parameter Estimation. *IFAC Proceedings Volumes*, 36(16):249–260, 2003. 5

[49] H. P. Wiendahl, H. A. ElMaraghy, P. Nyhuis, M. F. Zäh, H. H. Wiendahl, N. Duffie, and M. Brieke. Changeable Manufacturing-Classification, Design and Operation. *CIRP Annals*, 56(2):783–809, 2007. 1