
# OPTIMISATION
# IN
# MULTI-MODE SYSTEMS

Thesis submitted in accordance with the requirements of
the University of Liverpool for the degree of Doctor in Philosophy
by

**Mahmoud Mousa**

June 2018

# Contents

# Illustrations

## List of Figures

# List of Tables

# Notations

The following notations and abbreviations are found throughout this thesis:

| Symbol | Description |
|---|---|
| $\mathcal{A}$ | hybrid automaton |
| $\dot{X}$ | the first first derivative of the variables denoted by the set $X$ |
| $x_i$ | the ith variable that belongs to the set of variables $X$ |
| $V$ | control modes vertices |
| $E$ | control switches edges |
| $init(v)$ | initial condition predicate of a hybrid automaton $\mathcal{A}$ for control mode $v \in V$ |
| $inv(v)$ | invariant condition predicate of a hybrid automaton $\mathcal{A}$ for control mode $v \in V$ |
| $flow(v)$ | flow condition predicate of a hybrid automaton $\mathcal{A}$ for control mode $v \in V$ |
| $\Sigma$ | finite set of Events of a hybrid automaton $\mathcal{A}$ |
| $E$ | finite set of control switchs of a hybrid automaton $\mathcal{A}$ |
| $jump(e)$ | jump condition predicate for control switch $e \in E$ of a hybrid automaton $\mathcal{A}$ |
| $^\circ$C | degree celsius |
| $^\circ$C/$h$ | degree celsius per hour |
| $n$ | the total number of items in a knapsack problem |
| $w_i$ | the weight of the ith item in the input set of a knapsack problem |
| $p_i$ | the profit of the ith item in the input set of a knapsack problem |
| $c$ | the capacity of a knapsack problem |
| $\mathbb{Z}$ | the set of integer numbers |
| $\mathbb{Z}_+$ | the set of positive integer numbers |
| $\mathbb{R}$ | the set of real numbers |
| $\mathbb{R}_+$ | the set of positive real numbers |

| Symbol | Description |
|---|---|
| $Z_j(d)$ | the optimal solution value for the knapsack subset of $j \leq n$ items and a capacity of $d \leq c$ |
| $Z_{i-1}(d)$ | the optimal solution value for the knapsack subset of $j-1$ items and a capacity of $d \leq c$ |
| $Z^*$ | the overall optimal solution for a knapsack with set of $n$ items and a capacity of $c$ |
| $Z^G$ | the greedy approximation soluton for a knapsack problem |
| $OPT(x)$ | the optimal cost for the input $x$ |
| $\rho$ | the relative performance ratio |
| $\mathcal{O}$ | the complexity of |
| $\epsilon$ | the FPTAS precision value |
| $V(t)$ | the function describing the temperature in a room at time $t$ |
| $V_0$ | the initial temperature in a room |
| $V_{min}$ | the lower bound of the comfort zone |
| $V_{max}$ | the hidher bound of the comfort zone |
| $C$ | the thermal capacity of a room |
| kJ/K | kilojoules per kelvin |
| $Q$ | the heat input rate of a heater |
| kW | kilowatt |
| $\lambda$ | the thermal conductance between a room and the ambient air |
| kW/K | kilowatt per kelvin |
| $\mathbb{Q}_{>0}$ | the set of positive rational numbers |
| $\mathbb{Q}_{<0}$ | the set of negative rational numbers |
| $\mathbb{0}_N$ | an $N$-dimensional vector with all entries equal to 0 |
| $\mathbb{1}_N$ | an $N$-dimensional vector with all entries equal to 1 |
| $\|v\|$ | the $\infty$-norm for a vector $v$ (i.e. the maximum coordinate in $v$) |
| $M$ | finite set of modes |
| $m$ | a mode that belongs to $M$ |
| $N$ | number of continuous variables in the system (number of dimensions) |
| $A$ | the slope of all the variables in a given mode |
| $\pi_c$ | the cost per time unit spent in a given mode |
| $\pi_d$ | the cost of switching to a given mode |
| $\mathbb{Q}^N$ | a vector of $N$ rational numbers |
| $\mathbb{R}^N$ | a vector of $N$ natural numbers |

| Symbol | Description |
|---|---|
| $S$ | the safe set |
| $t_{max}$ | the horizotal time boundary |
| $£$ | Great Britain Pound |
| $£/h$ | pounds per hour |
| $\sigma$ | a schedule |
| $\pi(\sigma)$ | the total cost of the finite schedule $\sigma$ |
| $\pi_{avg}(\sigma)$ | the average cost of the infinite schedule $\sigma$ |
| $\Delta t_i$ | the time duration of a complete leap of type $i \in M$ |
| $\Delta \pi_i$ | the cost of a complete leap of type $i \in M$ |
| $\pi_e(i)$ | the effective continuous cost rate per time unit of using mode $i$ |
| $n_i \Delta t_i$ | all leaps, including the incomplete one |
| $o(n, v)$ | the minimal knapsack weight that yields value $v$ using subset of items $\{1, .., n\}$ where $v_n$ represents the item cost and $w_n$ is the item weight |
| $c^*$ | the $\alpha$-approximation solution |
| $o^*$ | the optimal solution |
| $V_\Sigma$ | the value of all items in a multiset |
| $M^+$ | the set of up modes |
| $M^-$ | the set of down modes |
| $M^0$ | the set of flat modes |
| $\Delta t_{m,m'}$ | the time duration of a leap of type $(m, m') \in M^+ \times M^-$ |
| $\Delta \pi_{m,m'}$ | the cost of a leap of type $(m, m') \in M^+ \times M^-$ |
| $\tau$ | sequence of timed actions |
| $\emptyset$ | empty set |
| $m^-$ | a mode that belongs to the down modes set |
| $V_{\text{end}}$ | the ending temperature of a finite schedule |
| $t_m^j$ | the time we use the mode $m$ in the $j^{th}$ switch in an abstracted schedule |
| $s_{max}$ | maximum number of discrete switchings in an abstracted schedule |
| $\overline{T}$ | the average execution time |
| $\overline{e}$ | the average error |

| Abbreviations | Description |
|---|---|
| HVAC | Heating, Ventilation and Air-conditioning system |
| FPTAS | fully polynomial time approximation scheme |
| TRNSYS | TRaNsient SYstems Simulation Program |
| IBPT | International Building Physics Toolbox |
| min | minimise |
| max | maximise |
| KP | Knapsack Problem |
| UKP | Unbounded Knapsack Problem |
| PTAS | Polynomial-time approximation scheme |
| TM | Turing machine |
| PTIME | polynomial time |
| NP | non-deterministic polynomial time |
| NEXPTIME | non-deterministic exponential time |
| Pspace | polynomial space |
| LogSpace | logarithmic space |
| BMS | bounded rate multi-mode systems |
| MPC | model predictive control |
| LP | linear programming |
| ILP | integer linear programming |
| poly | Polynomially |

# Preface

This thesis expands and extends two published conference papers. These papers are

1. Mahmoud A. A. Mousa, Sven Schewe, and Dominik Wojtczak. Optimal Control for Simple Linear Hybrid Systems. In *Proc. of TIME*, pages 12–20. IEEE Computer Society, 2016.

2. Mahmoud A. A. Mousa, Sven Schewe, and Dominik Wojtczak. Optimal Control for Multi-Mode Systems with Discrete Costs. In *Proc. of FORMATS*, pages 77-96. Springer, 2017.

We have been invited to submit an extended version of the TIME's paper into a special issue of a high ranked journal *"Journal of Logical and Algebraic Methods in Programming"*. The journal paper is still under review. We're writing two other papers. The first one is a tool paper that describes the tool we have designed to produce approximation solutions for the optimisation in the multi-mode multi-dimension systems with discrete costs. The other paper is an overall journal paper that studies all the cases of optimisation in Multi-mode systems with discrete costs.

# Abstract

We study cost optimisation in multi-mode systems with discrete costs. We first solve the problem in one dimension and next we study it in multiple dimensions. As a motivating example, we study the temperature control in buildings using heating, ventilation and air-conditioning system HVAC while paying the minimal cost as possible. By optimising the behaviour of the HVAC systems, lots of energy could be saved. We are interested in finding optimal solutions as well as approximate solutions with guarantees.

The optimisation problem in one dimensional multi-mode systems with discrete costs –which is a simple subclass of linear hybrid systems– consists of one continuous variable and global constraints. Each state has a continuous cost attached to it, which is linear in the sojourn time, while a discrete cost is attached to each transition taken. We study the cost minimisation for multi-mode single dimension system with and without an *idle* mode. We show the corresponding decision problem with finite time horizon to be NP-complete while the infinite time horizon problem is in LogSpace. We search for optimal safe schedules if the safe region is defined by the hyperrectangle bounded by $V_{min}$ and $V_{max}$. For the optimisation problem with an idle mode which is a cost free mode where all heaters are turned off, we can use the *idle* mode to decrease the room's temperature and can switch heaters on to increase the room's temperature. This implies that we pay two types of costs (discrete and continuous) to heat the room up using heaters and pay nothing to cool it down using the idle mode. We present a pattern of the optimal schedules that has to end with $V_{min}$ and contains the lowest number of switches between a heating mode and an idle mode. We use this pattern to model the problem using the integer linear programming and hence provide an optimal solution. We also develop a two-approximation algorithm as well as an FPTAS to find an approximate solution. We provide a Java implementation that finds an optimal solution as well as an approximate one. We show that, for multi-mode systems with small number of modes, the optimal integer programming algorithm as well as the FPTAS approximation algorithm run quickly and give the exact optimal schedule or one which is very close to optimal (in the case of using the

FPTAS approximation algorithm), respectively. In all other instances, the constant factor approximation algorithm is the best choice, as it runs really quickly and most of the time gives a near-optimal solution.

We study also the optimisation problem in multi-mode single dimension systems without the idle mode where we may use coolers/air-conditioners to cool the room down. So, any mode is allowed and all the modes may have costs. Now, for the system without the idle mode, we show that its infinite time horizon version is still in LogSpace. For its finite time horizon version, we show that the pattern for the optimal finite schedules that was introduced in the idle mode systems is not producing the minimal cost. We show that the pattern of an optimal schedule can be one out of 44 possibilities. We present cost non-increasing operations that transform any safe schedule to the optimal shape. Based on this, we propose FPTAS approximation as well as a 3-approximation algorithm that runs in $\mathcal{O}(|\mathcal{A}|^7)$ time. We prove that the optimisation problem still NP-hard.

Then, we study optimal control in multi-mode multi-dimension systems with discrete costs. We prove that the optimal safe schedule may not exist. This problem may occur if the initial point lies on the safe boundaries and there is no mode that can be used to get rid of the boundary and preserves the safety constrains. We present a solution for this problem by permitting $\epsilon$-safety deviation where the safety boundaries are extended from $V_{min}$ and $V_{max}$ to $(V_{min} - \epsilon)$ and $(V_{max} + \epsilon)$. We show that if a limit-safe abstract schedule exists in $\mathcal{A}$, then there exists one of exponential length and it can be constructed in polynomial time. We also show that finding an optimal limit-safe abstract schedule in $\mathcal{A}$ can be done in nondeterministic exponential time. Next, we show that if a limit-safe abstract schedule exists, then finding an $\epsilon$-safe $\epsilon$-optimal strategy can be done in deterministic polynomial space and propose an algorithm to find it. We present $\epsilon$-safe $\epsilon$-optimal approximation algorithm that permits adding as many timed actions for modes that do not have discrete costs while limiting the number of timed actions that use modes with discrete costs.

# Acknowledgements

I would like to thank my supervisors Dominik Wojtczak and Sven Schewe for their help and guidance along my study.

I would like to thank my parents and all my family for their support and encouragement.

I would like to thank my small family: my wife Esraa, my daughter Lara and my little baby Tmara for giving my life a meaning and without them I would not be able to continue fighting for better life.

I would like to thank my examiners for accepting to evaluate my thesis.

# Chapter 1

# Introduction

This chapter introduces the optimisation problem in multi-mode systems with discrete costs which are a special kind of hybrid systems. This problem is motivated by the minimisation of the average cost being paid in the indoor temperature control systems using HVAC systems. We show how to find for them optimal solutions and approximate solutions with guarantees. Since HVAC systems account for about 50% of the total energy cost in buildings [48], a lot of energy can be saved by optimising their control. Next, we briefly mention the related work in the area of hybrid systems, multi-mode systems (specially regarding the reachability problem and its decidability) and the minimisation of energy usage in temperature control systems. Finally, the main contributions as well as a summary for each chapter is given.

## 1.1 Hybrid Systems

Hybrid systems [4, 5, 29] are systems that contain both discrete and continuous behaviour. The discrete part represents the states and the continuous part is the switching behaviour between these states. Then switching behaviour depends on the system dynamics and in most cases is usually an ordinary differential equation. In our work, we study a special case of hybrid systems which is the linear hybrid systems where the switching behaviour is represented by linear functions. Hybrid systems can be modeled as hybrid automata [5] with variables that change its values continuously with time.

### 1.1.1 Hybrid Automata

As shown by Alur-Henzinger in [5], a hybrid automaton $\mathcal{A}$ is shown as follows. (i) n-real variables $X = \{x_1 \dots x_n\}$ where $n$ represents the number of variables or in

other words it refers to the number of dimensions in the multi-dimension system. The first derivative of the variables denoted using the dotted form $\dot{X} = \{\dot{x_1} \dots \dot{x_n}\}$ while the prime form $X' = \{x'_1 \dots x'_n\}$ represents the discrete values of the variables in which the switches take place. (ii) A finite multigraph $(V, E)$ of control modes vertices $V$ and control switches edges $E$. (iii) A finite set of events.

For the multigraph, there exist three labeling functions *init(v)*, *inv(v)*, and *flow(v)* for each control mode $v \in V$. The first two functions *init(v)* and *inv(v)* represent the initial condition predicate and the invariant condition predicate respectively where those free variables belong to the set $X$. The last function *flow(v)* is the flow condition predicate with free variables belong to the union between the continuous variables and its discrete values at the switches $X \cup X'$. For each control switch $e$ there exist a labeling function for edges *jump(e)* which represents the jump condition predicate with free variables from $X \cup X'$. For each control switch $e \in E$ there exist a function that assign an event from a finite set of Events $\Sigma$ to the control switch using an edge labeling function event: $E \to \Sigma$.

Figure 1.1 shows an example of a hybrid automaton with one continuous variable and ordinary differential equation transition behaviour. It represent a temperature control system where there exists a heater that can be either on or off. The system has one continuous variable $x$ which represents the room temperature and the aim is to maintain the room temperature between 18°C and 22°C. The temperature of the room is modeled by $\dot{x} = 5 - 0.1x$ when the heater is on. When the heater is turned off, the temperature inside the room changes as follows $\dot{x} = -0.1x$. The invariant conditions are $x \geq 18$ and $x \leq 22$°C for the *off* and *on* states, respectively. The guard transition from the *off* state to the *on* state is $x < 19$°C while it is $x > 21$°C the other way around. The initial state shown is $x = 20$°C.

Figure 1.2 shows another example for the temperature control system explained in Figure 1.1 but with a linear transition behaviour where the room will be heated with the rate of $4°C/h$ when the heater is on and cooled with the rate of $2°C/h$ when the heater is off. This is an example of a linear hybrid system.

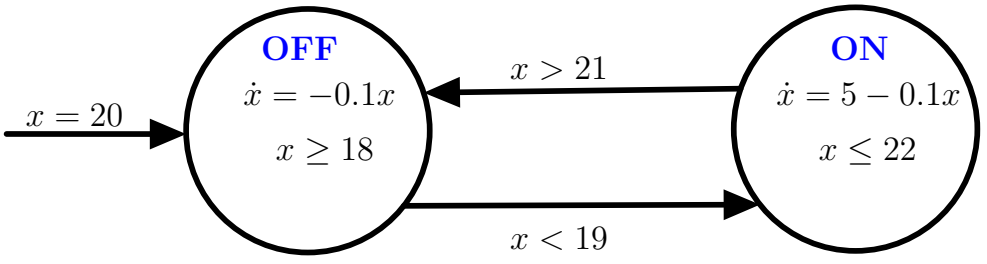



FIGURE 1.1: Hybrid system example.

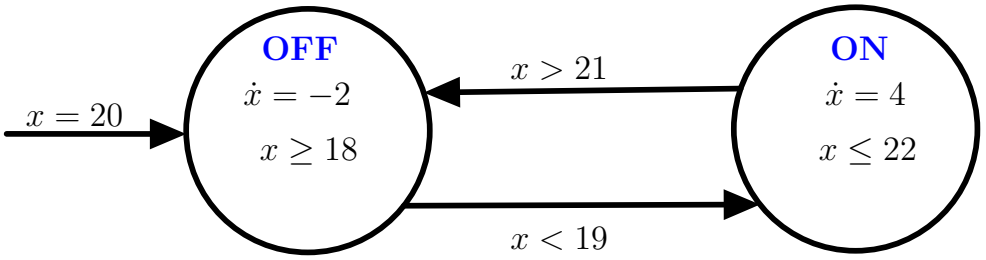



FIGURE 1.2: Linear hybrid system example.

A particular subclass of hybrid automata is timed automata for which all the variables have slope equal to 1 [6].

### 1.1.2 Multi-mode Systems

Multi-mode systems [9] are an important subclass of linear hybrid systems [4], which consist of multiple continuous variables and global invariants for the values that each variable is allowed to take during a run of the system. However, unlike for the full linear hybrid systems model, multi-mode systems have no guards on transitions and no local invariants. Examples of the multi-mode systems can be seen as the systems that switch between finite number of operations such as transistors or diodes (in electric systems) and switches (in hydraulic systems).

## 1.2 Problem Statement

We study multi-mode systems with discrete costs, which extend linear hybrid systems by adding both continuous and discrete costs to states. Every time a transition is taken (i.e. when the current state changes), the discrete cost assigned to the target state is incurred. The continuous cost is the sum of the products of the sojourn time in each state and the cost assigned to this state. Our aim is to minimise the total cost over a finite-time horizon or a long-time average cost over an infinite time horizon. The formal definition of all the terms are shown later in Section 2.5.

### 1.2.1 Motivation

The motivation behind this work is to achieve the optimal control of heating, ventilation, and air-conditioning (HVAC) systems. HVAC systems account for about 50% of the total energy cost in buildings [48], so a lot of energy can be saved by optimising their control. Many simulation programs have been developed to analyse the influence of control on the performance of HVAC system components

such as TRNSYS [3], EnergyPlus [1], and the Matlab's IBPT [2]. Our work has the advantage over the existing control theory techniques that it provides guarantees. Although the actual dynamics of a HVAC system is governed by linear differential equations, one can argue [41, 43, 45] that constant rate dynamic, as in our model, can approximate well such a behaviour. An example of such a model is shown in Figure 1.2

### 1.2.2 Motivating Example

We optimise the usage of the HVAC systems to maintain the temperature of a room –in single dimension system– or a finite number of rooms –in the multi-dimension system– within a safe temperature range(s). We assume that we have $n$ heating/cooling modes. The heating modes correspond to the heaters behaviours while the cooling modes reflect the idle mode or the air conditioners behaviours. Each mode has an initial cost (discrete) which is paid every time we use the mode and a running cost per unit time (continuous) which is paid as long as we use the mode. We assume that the heating behaviours as well as the cooling behaviours are linear.

We study the problem over finite and infinite time horizon. We are interested in finding optimal safe schedules (if exist). We also find approximate solutions with guarantees. We study the problem in theory as well as in practical. We design a tool using Java and Gurobi that implements the suggested algorithms.

The main challenging problem is how to provide optimal safe schedules for systems with finite/infinite time horizon. We prove in Section 3.3 that this scheduling problem is NP-complete which means that it is unlikely to be solved in polynomial time. The way we prove it is by reducing this problem to the well known knapsack problem. So, we introduce the main concept of the knapsack problem in the Chapter 2.

## 1.3 Optimal Solution

We study optimal algorithms for the total cost minimisation problem in multi-mode systems. As we will show in Section 3.3, the optimal solution can be found in exponential time. Approximation algorithms with guarantees can be designed so that they produce approximation solutions in polynomial time.

## 1.4 Approximate Solution

We study approximation algorithms for the total cost minimisation problem in multi-mode systems. We say that an algorithm is a *constant factor approximation algorithm* with a *relative performance* $\rho$ iff, for all inputs $x$, the cost of the solution that it computes, $f(x)$, satisfies $OPT(x) \leq f(x) \leq (1+\rho) \cdot OPT(x)$, where $OPT(x)$ is the optimal cost for the input $x$.

We are particularly interested in polynomial-time approximation algorithms. A polynomial-time approximation scheme (PTAS) is an algorithm that, for every $\rho > 0$, runs in polynomial-time and has relative performance $\rho$. Note that the running time of a PTAS may depend in an arbitrary way on $\rho$. Therefore, we typically strive to find a fully polynomial-time approximation scheme (FPTAS), which is an algorithm that runs in polynomial-time in the size of the input and $1/\rho$. The 0-1 Knapsack problem is a well-known optimisations problem, which possesses multiple FPTASes (see e.g. [32]).

## 1.5 Related Work

Our model can be viewed as weighted extension of the linear hybrid automata model in [5, 29], but with global constraints.

In [5], Alur and Dill presented the theory of timed automata and how this can be used to address the time behaviour of real time systems. They showed nice properties for timed automata. For example, the time languages recognise the non deterministic timed automata are closed under union and intersection but not under complementation. On the other hand, the time languages recognise the deterministic timed automata are closed under all the boolean operations.

The theory of hybrid automata was presented in [29]. Henzinger presented the definition of hybrid automata which model real-time systems that contain discrete behaviours. The paper demonstrated a temperature control example and showed how this can be modelled as hybrid automaton. The composition of hybrid automata was presented using multiplication of hybrid automata systems. The author studied four main problems to achieve safety and liveness requirements. These problems are the reachability problem, the emptiness problem, the time trace inclusion problem and the time-abstract trace inclusion problem.

Even basic questions for the general linear hybrid automata model are undecidable already for three variables and not known to be decidable for two variables [11]. Most of the research for this model has focused on qualitative objectives such as reachability. For hybrid systems, the reachability problem is one of the

most important problems that has been studied for the recent two decades until now. It was shown before in [6] that the reachability problem for hybrid systems is undecidable even for its very simple class known as linear hybrid systems. From that time, researchers started studying sub-classes of hybrid systems with special specifications and constraints that result in decidability. Most of these sub-classes are still undecidable while other few sub-classes were shown to be decidable for special constraints types. At the same time, these sub-classes became undecidable if small changes are applied in the constraints or the continuous variables. Various subclasses of hybrid systems with a decidable reachability problem were considered, see e.g. [11] for an overview. In particular, reachability in linear hybrid systems, where the derivative of each variable in each state is constant, can be shown to be decidable for one continuous variable by using the techniques from [35]. In [6], it has been shown that reachability is decidable for timed automata.

In [9], Alur, Trivedi and Wojtczak studied the optimal scheduling for constant-rate multi-mode systems where for every mode $m$ and variable $x_i$, the value of the variable $x_i$ increases by $C_i^m \cdot t$ times after spending some time $t$ in the mode $m$ where $C_i^m$ represents the constant rate change of the variable $x_i$ while using the mode $m$. They studied the safe schedulability and the safe reachability problems by devising polynomial time algorithms to solve them. The work was extended to solve the optimal schedulability problem of minimising the average cost of systems with only continuous cost part for every mode but with no switching costs and only for the infinite time horizon. The optimal schedulability problem with reachability cost objective was proved to be decidable. This was done by proposing polynomial time scheduling algorithms to find the schedules. Adding other constraints, except the global ones, or guards with the mode switches makes the problem undecidable. They studied the energy optimisation problem for HVAC systems as an application. This was done with only continuous costs that are being paid per unit time in multiple zones (dimensions). The aim was about minimising the average cost being paid while keeping the temperature of the rooms within a comfort zone.

In [7, 8], Alur et al. studied the schedulability problem for bounded rate multi-mode systems (BMS). The system permits free switches between the system modes. Each mode is specified by a mode-dependant rates vector. The paper presented the schedulability problem as a two-players game between the scheduler and the environment. The game rule is that the scheduler suggests a mode and a time while the other player (environment) selects the allowed mode rate which changes the system sate within the safe set. The safe set was shown to be a closed convex polytope. The problem of finding non-Zeno schedules for any arbitrary starting state was proved to be decidable by introducing a winning strategy. Also,

the schedulability problem for the BMS was proved to be co-NP complete in general and for the systems with only two variables can be solved in polynomial time. In [13], the same nice theorems and algorithms regarding the safe reachability problem and its decidability were presented for also the bounded rate multi-mode systems but this time it is applied to the robotic motion planning problems.

In [52], the optimal control for linear-rate multi-mode systems was studied by Wojtczak. He studied a hybrid automaton model, where the dynamics are governed by linear differential equations, but without switching costs and only with an infinite time horizon. The author studied how power can be minimised in system that use all the HVAC components to keep the rooms' temperature inside a building within a comfort zone. This paper studied the existence of safe controllers that produce safe schedules with certain reachability and optimal objectives. The author proposed polynomial time algorithms for producing safe schedules given initial state within the interior set of the safe region. The optimisation problem with continuous costs was also studied and as a result, the author suggested polynomial time algorithms for finding the schedules with minimum average cost, peak demand cost and a weighted sum between them. This papers shows that, for any number of variables, a schedule with the optimal long-time average cost can be computed in polynomial time.

In [15], long-time average and total cost games have been shown to be decidable for hybrid automata with strong resets, in which all variables are reset to 0 after each discrete transition. The long-time average and total cost optimisation for the weighted timed automata model have been shown to be PSpace-complete (see e.g. [14] for an overview). In [17], Brihaye et. al. studied the reachability problem for hybrid automata over a bounded time which has a rational value. It was shown that for rectangular hybrid automata that have only positive rates, the problem is decidable while it is undecidable if both positive and negative rates are allowed.

In [23], optimising the energy consumption in buildings was presented by using statistical hybrid automata with the help of the UPPAAL [18] model checker. In [36], Larsen et al. studied the statistical hybrid switched systems. As an application, they suggested an online synthesis method for controlling the floor temperature inside a house with multiple rooms. The reason of using online synthesize instead of the off-line is that controlling a continuous variable results in an uncountable state space, which is impossible to be explored, and even after digitization, the sate space still huge and requires tons of searches to be explored. The authors used the learning methods provided by the UPPAAL Stratego [24] for small-scales systems while with industrial-scales models, the short term control strategies are applied iteratively while the author suggested a compositional

methodology to combine them. UPPAAL Stratego [24] supports the analysis of the expected cost in linear hybrid systems, but uses a stochastic semantics of these models [23, 25]. I.e. a control strategy induces a stochastic model where the time delay in each state is uniformly or exponentially distributed. This is different to the standard nondeterministic interpretation of the model, which we use in our work.

In [43], Nghiem et al. studied the green scheduling for aggregate peak power reduction for systems with multiple zones. The authors didn't consider any thermal interactions between the building's zones. The system behaviour was modelled by ordinary differential equation. A solution for the aggregate peak reduction problem was suggested using combinatorial optimisation to minimise the peak constraint while ensuring the satisfaction of the safety constraints. Also, the lazy scheduling was introduced to solve peak demand reduction problem by doing the switching decisions only at the thresholds which make it efficient and scalable for the systems with large number of modes. The same authors in [44] continued the peak power demand reduction problem by providing a more accurate modelling for the radiant systems. They extended the work to electric radiant floor heating systems. They generated periodic schedules as a result of the reduction problem. A simulation was done in EnergyPlus for small-scale systems and Matlab for large-scale systems. In [45], the work was extended again to address the effect of the disturbances and being more appropriate for real applications. As a result of taking the disturbance into account, periodic schedules can not be generated any more and the authors presented an online state feedback scheduling strategy to generate online schedules. The effectiveness measurements that were done for the strategy using Matlab over a hydronic radiant system with 10-zones showed a peak demand reduction ratio of 77.8% and a total energy consumption reduction of 31.2%.

The peak electricity reduction was also studied in [46]. The authors used the model predictive control (MPC) and real time pricing to reduce the peak electricity demand in building climate control. In [37], On-Off optimal control was considered for air conditioning and refrigeration. The drawback of using MPC is its high computational complexity and the fact that it cannot provide any worst-case guarantees.

## 1.6 Contributions

In [41], we proved that the optimisation in a single room system over a finite time horizon is NP-Hard problem by having a reduction from a one dimension knapsack

problem which is known to be NP-hard. We studied a special case of the optimal control for multi-mode systems with discrete cost in a single room with a simplifying assumption. We assumed that an idle mode always exists to cool the room down and it is cost free. We found optimal solution by modelling the system as an integer programming problem. We also provided a constant factor approximation algorithm (Two approximation) as well as a fully polynomial-time approximation (FPTAS) algorithm to find approximation solutions with guarantees. We implemented all the algorithms presented in [41] using Matlab software in order to run some experiments. We run experiments over the knapsack strongly correlated hard instances. The tests showed that the two approximation algorithm has the fastest running time among the algorithms I suggest in Chapter 3 and always give an answer and terminate. The FPTAS approximation suffers from time-out problem especially when we consider hard problems with large time horizons and high number of modes (heaters). We also found that the integer programming algorithm crashes due to memory management issues.

Later we submitted an extended journal version to study the same problem in [41]. In this paper, we presented a fast FPTAS approximation algorithm. This approximation algorithm uses a reduction to a 0-1 knapsack problem and an optimal dynamic programming algorithm to find solutions with guarantees with a short running time. We also reimplemented all the algorithms in Java to solve the crashing problem mentioned in [41]. We provided more tests with more datasets and results to compare between the algorithms.

In [42], we extended the work done in [41] while dropping the simplifying assumption about the existence of an idle mode and generalised the model to multiple dimensions. We studied multi-mode multiple dimensional systems with discrete costs, which extend linear hybrid systems by adding both continuous and discrete costs to states. We considered a motivating example of controlling the temperature in multiple rooms simultaneously using heaters and air conditioners. In such a scenario, we might have different pleasant temperature ranges in different rooms and the temperatures of the individual rooms may influence each other. Naturally, controlling a multi-dimensional multi-mode systems is more complex than controlling a one-dimensional multi-mode system. We showed that the optimal schedule may not exist. We developed a nondeterministic exponential time algorithm for the construction of optimal control, whose complexity is only driven by potentially required high precision in exponentially many mode switches. We showed that allowing for an $\varepsilon$-deviation from the ranges of pleasant temperatures reduces the complexity to PSpace. We provided more detailed analysis of the one-dimensional setting, we showed that an optimal schedule always exist by defining

transformation operations that transform any schedule to the optimal one. We managed to prove similar nice algorithmic properties as in [41], i.e. the existence of finitely many patterns for optimal schedules, polynomial constant-factor approximation algorithm and an FPTAS. However, as opposed to the existence of a unique pattern for an optimal schedule in [41], we showed that that there can be 44 different patterns when the simplifying assumption is dropped. Also, we used the properties of the optimal schedules we proved to presented a constant-factor approximation algorithm that requires $\mathcal{O}(n^7)$ complexity, while in [41] it sufficed to use one mode all the time and the algorithm ran in linear time.

Regarding the optimal control for multi-mode multi-dimension systems with discrete costs as well as the continuous costs, we devised a $\rho$-approximate $\epsilon$-safe algorithm which uses the idea behind the greedy approximation technique.

## 1.7 Thesis Outline

- Chapter 2 contains a background information about knapsack problems and complexity classes. It also contains the modelling of HVAC system as an ordinary differential equation and the approximation of that model into a linear one. The formal definition of the system as linear hybrid system is also presented. It also introduces definitions of the terms that will be used along the thesis.

- In Chapter 3, we study the optimisation in Multi-mode single dimension systems with simplifying assumption. The optimisation problem in multi mode systems would be simple if it is permitted to use a cost free idle mode to cool the room down without paying money. We prove that the optimisation problem is NP-hard by reduction from knapsack problem. We introduce one of the optimal schedule forms. We use the optimal schedule form we proved to design optimal algorithms using integer programming and dynamic programming concepts. We also introduce the suggested approximation algorithms with guarantees such as two approximation and FPTAS approximation algorithms.

- In Chapter 4, we study the optimisation in Multi-mode single dimension systems without the simplifying assumption. We show the optimal schedule form by introducing operations that transform any schedule to the optimal one. We introduce the optimal algorithm and approximation algorithms with guarantees –three approximation and FPTAS approximation– to solve the

optimisation problem for the general case without the simplifying assumption.

- In Chapter 5, we study the optimisation in multi-mode multi-dimension systems. We prove by an example that the optimal schedule may not exist. The chapter also contains the non-deterministic exponential time algorithm for the construction of optimal control and the approximation algorithm when $\epsilon$ deviation is enabled.

- In Chapter 6, We test the algorithms presented in Chapter 3 only for the multi-mode single dimension systems with a simplifying assumption. We do tests for normal and hard knapsack instances. We compare between the optimal and approximation algorithms with respect to the average execution time and the percentage error for different situations.

- In Chapter 7, we conclude our work and discuss the future work.

# Chapter 2

# Preliminaries

As the minimisation problem we study can be reduced to a Knapsack problem which will be shown in Chapter 3, this chapter introduces the knapsack problem. We presents different types of the Knapsack problem as well as different algorithms to solve them by either producing the optimal solution or an approximate one. The algorithms' complexity classes are also presented in Section 2.2.

Furthermore, Section 2.3 introduces the model of the motivating example we used for the multi-mode systems with continuous and discrete cost. The example being addressed studies optimising the usage of heating ventilation and air-conditioning (HVAC) systems while maintaining the temperature of the room(s) within a comfort zone. We are interested in the heating behaviour of the HVAC systems. So, the dynamics of the system are modelled as an ordinary differential equation. For narrow comfort zones, it is sufficient to use linear approximations instead of using linear differential equations. Hence, we introduce a behaviour linearisation which describes the room temperature inside the comfort zone. Now, the room temperature is described by a linear (continuous) equation bounded by two discrete values (the comfort zone boundaries) which can be formally defined as linear hybrid system as shown in Section 2.4.

In Section 2.5, we present main terms and their definitions –supported by demonstration examples– that are used to express schedules. The main concepts for calculating the schedule's average cost are also shown.

## 2.1 Knapsack Problem

### 2.1.1 Introduction to Knapsack Problem

Knapsack problem has been studied for solving the optimisation problems with maximisation desires. The knapsack problem definition was introduced in 1896 by

Mathews in [39]. Mathews showed that the knapsack problem is solvable by doing a reduction from the integer linear programming so, he showed that the Knapsack problem is the same hardness as the integer programming problems. This was done by combining several constraints together into one constraint.

Knapsack problem can be applied to financial and marketing problems. Consider daily-life situation in which a person is doing shopping and the aim is to buy the largest number of items from a list he/she can while being constrained by a budget limit. So, the person wants to maximise the number of items he/she could buy (called the objective) while not exceeding the budget limit (called the constraint). This can be represented as a 0-1 Knapsack problem such that the objective function is

$$\max \sum_{i \leq n} x_i$$

where $x_i \in \{0, 1\}$ represents whether the item $i$ is selected (if $x_i = 1$) or not – otherwise– and $n \in \mathbb{Z}_+$ is the number of the items inside the list and the constrain is

$$\sum_{i \leq n} x_i w_i \leq c$$

where $c > 0$ is the budget limit and $w_i$ represents the price of item $i$. Such instances can be solved using integer programming [22, 49].

### 2.1.2 Types of Knapsack Problems

Suppose that we have $n$ items where each item $j \leq n$ has a weight of $w_j > 0$ and a profit $p_j \geq 0$. The aim is to pack a subset of the set $N = \{1, \ldots, n\}$ inside a knapsack with capacity value $c$ that maximises the profit. The knapsack problem can be classified according to the decision problem into *binary decision knapsack problem, linear decision knapsack problem* and *quadratic decision knapsack problem*. In our work, we are interested only in binary and linear decision problems. A short descriptions are now presented for these two decision problems [32].

#### Binary Decision Knapsack Problem

The simplest decision with the knapsack problem is the binary decision. It is known also as 0-1 knapsack problem. Each item $j$ has a decision variable $x_j \in \{0, 1\}$. When $x_j = 1$, it means that the item $j$ will be added to the knapsack. On the other hand, when $x_j = 0$, it means that the item $j$ will be rejected as there are better alternatives to be added instead. The problem can be modeled as integer programming problem with two decision values 0 or 1 as follows:

$$maximise \qquad \sum_{j=1}^{n} p_j x_j \tag{2.1}$$

$$subject\,to \qquad \sum_{j=1}^{n} w_j x_j \leq c \tag{2.2}$$

$$x_j \in \{0,1\}, \quad j = 1,\ldots,n \tag{2.3}$$

The equation 2.1 contains the objective function $\sum_{j=1}^{n} p_j x_j$ with a maximisation desire while the equation 2.2 represents the constraint that does not allow the solution to exceed the capacity $c$.

#### Linear Decision Knapsack Problem

The knapsack problem can also be modelled as linear decision problem where the decision variables shown in equation 2.3 are as follows.

$$x_j \geq 0,\, x_j \in \mathbb{R}_+, \quad j = 1,\ldots,n \tag{2.4}$$

If the $x_j$ variables are more to be positive integers (i.e. $x_j \in \mathbb{Z}_+$), then the knapsack problem is called *unbounded knapsack problem (UKP)* and if variables are bounded ($x_j \in \{0,\ldots,m\}, m \in \mathbb{Z}_+$), then the problem is called the *bounded knapsack problem (BKP)*.

#### Knapsack Problem in Multiple Dimensions

The decision knapsack problem is defined in k-dimensional system as maximisation of an function with $k-$ linear constrains. The simplest form is shown as follows:

$$\begin{aligned} Maximise \qquad & \sum_{j=1}^{n} p_j x_j \\ Subject\,to \qquad & \sum_{j=1}^{n} w_{ij} x_j \leq c_i\,, \forall i \leq k, x_j \in \{0,1\}, j = 1,\ldots,n \end{aligned} \tag{2.5}$$

### 2.1.3 Knapsack Problem Algorithms

The decision version of the knapsack problem is NP-complete while its optimisation version which searches for the exactly for the optimal solution is NP-hard [31]. So, it is unlikely that there is a polynomial time algorithm that gives the

optimal solution for the knapsack problem. The optimal solution can be computed by formulating the knapsack problem as a linear programming problem and solving it using one of the well known optimisers such as Gurobi[28], CPLEX [38] or the optimisation toolbox provided by MATLAB [16]. In this section, we introduce the dynamic programming based algorithm to solve the knapsack problem optimally. We also introduce the greedy approximation algorithm that produces approximation solutions in polynomial time.

### Dynamic Programming Algorithm

The concept of dynamic programming [12, 32] is to solve the required problem over small subsets of the input datasets and iteratively extend the solution step by step to consider all the dataset and hence produce the global optimal solution. The dynamic programming concept can be applied to the Knapsack optimisation problem by starting with the case of finding an optimal set that can be selected from initial input set that contains only one item. In this case, the optimal solution is trivial. The optimal set contains the item if its weight is smaller than or equal to the problem capacity (i.e. the item is fitted into the knapsack) while it contains nothing otherwise. Iteratively, new items are added and the new optimal set is computed based on the knowledge of the old optimal set and the details about the new added item. As long as there is an item left to be considered, the algorithm checks whether the latest optimal value can be enhanced or not by using the new item. The algorithm keeps adding items until all the items are considered and hence the algorithm terminates and returns the optimal solution.

To do that, let us define sub-problems of the main Knapsack problem that consist the set of item $\{1, \ldots, i\}$ and capacity of $0 \leq d \leq c$ which means that we are going to solve the optimisation problem over smaller item sets constrained by smaller capacities. This can be seen as a two dimension array of $n$ columns, where the column index corresponds to the number of items, and $c$ rows ,where the row index represents the capacity. A cell indexed by $(x, y)$ inside the two dimension array contains the optimal solution for the knapsack problem over the first $y$ items from the overall item set constrained by the capacity equal to $x$.

Let $Z_j(d)$ be the optimal solution value for subset of $j$ items while $Z_{j-1}(d)$ is the optimal solution value for the subset of $(j-1)$ items for a knapsack of capacity $d$. Initially, $Z_{j-1}(d)$ is initialized to zeros for all capacities $0 \leq d \leq c$. The optimal solution value $Z_j(d)$ can be computed as shown in the following recursive equation

while the algorithm is presented in Algorithm 1 with a complexity of $\mathcal{O}(cn)$.

$$Z_j(d) = \begin{cases} Z_{j-1}(d) & \text{if } d < w_j \\ max\{Z_{j-1}(d), Z_{j-1}(d - w_j) + p_j\} & \text{if } d \geq w_j \end{cases}$$

**Algorithm 1** Dynamic programming algorithm computing the optimal solution for the 0-1 Knapsack problem [32].

**Input:** Set of items $\{1, \ldots, n\}$, items' weights and profits $w_j$ and $p_j$, respectively where $j \in \{1, \ldots, n\}$ and the budget limit $c$.
**Output:** The optimal solution $Z^*$.

```
for d := 0 to c do
    Z_0(d) = 0
end for
for j := 1 to n do
    for d := 0 to w_{j-1} do
        Z_j(d) = Z_{j-1}(d)
    end for
    for d := w_j to c do
        if Z_{j-1}(d - w_j) + p_j > Z_{j-1}(d) then
            Z_j(d) = Z_{j-1}(d - w_j) + p_j
        else
            Z_j(d) = Z_{j-1}(d)
        end if
    end for
end for
return the overall optimal solution Z* = Z_n(c).
```

### Greedy Algorithm

The idea for a greedy algorithm is to pack the items with the highest profit per weight which means that we select the most precious items that can be fitted into small space. The greedy algorithm generates an approximate solution in polynomial time. The greedy algorithm for the 0-1 Knapsack problem is shown by Algorithm 2. A pre-processing stage is to sort the items according to the profit per weight value such that

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \ldots \geq \frac{p_n}{w_n}$$

and start packing items with the highest profit per weight values. The packing operation is done item by item and after the addition of every item, we calculate the remaining knapsack capacity by subtracting the item weight from the original/latest capacity. The item can be packed if and only if it can be fitted inside

the knapsack which means that its weight is smaller than the remaining capacity.

**Algorithm 2** Greedy algorithm for the 0-1 Knapsack problem [32].

**Input:** Set of items $\{1, \ldots, n\}$, items' weights and profits $w_j$ and $p_j$, respectively where $j \in \{1, \ldots, n\}$ and the budget limit $c$.
**Output:** The approximate solution $Z^G$.

```
1: w̄ := 0;  Z^G := 0;
2: for j := 1 to n do
3:     if w̄ + w_j ≤ c then
4:         x_j := 1
5:         w̄ := w̄ + w_j
6:         Z^G := Z^G + p_j
7:     else
8:         x_j := 0
9:     end if
10: end for
11: return the solution Z^G corresponding to the item set {j|x_j = 1}.
```

**Example 2.1.** *Suppose that we are given a knapsack problem with three items. The weight and profit for each item are as follows* $\{(2,6),(3,8),(1,1)\}$. *The items are sorted according to the profit per weight values as the values of the profit per weight are* $3, 2\frac{2}{3}$ *and* 1, *respectively. If the knapsack capacity is* $c = 4$, *the optimal value computed from the run of the dynamic programming method shown in Table 2.1 is* 9 *by selecting the second and the third items while the greedy algorithm selects the first and the third items with a total profit of* 7. *After selecting the first item, the remaining knapsack capacity is* $4 - 2 = 2$ *which is not enough to pack the second item with a weight of* 3.

## 2.2 Complexity Classes

The complexity class is the set of problems that are solved by $\mathcal{O}(f(n))$ of resource $R$ using an abstract machine $M$ where n represents the input size dataset of the a given problem. The complexity classes study the rate of growth of the resources needed if the input size $n$ is increased. If the execution of the algorithm does not depend on the input size $n$, the value $f(n)$ is constant while it can be polynomial, exponential or logarithmic for other algorithms that depend on the input size. The resource $R$ can be time/CPU (for time complexity) or memory (for space complexity). So, we are going to briefly present some of the complexity classes we will mention during the thesis.

| d \j | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 2 | 0 | 6 | 6 | 6 |
| 3 | 0 | 6 | 8 | 8 |
| 4 | 0 | 6 | 8 | **9** |

TABLE 2.1: Dynamic programming solution for the Knapsack problem with $n = 3$ items and capacity $c = 4$. The first row represents the number of items while the first column denotes the capacities for each knapsack sub-problem.

### 2.2.1 Time Complexity Classes

- **P** or **PTIME**. It stands for polynomial time complexity class. It is the complexity class that contains the decision problems which are solvable by a deterministic Turing machine (TM) in polynomial time.

  *Example.* Finding the greatest common factor of two integer inputs $gcd(x, y)$ is in P-complexity class.

- **NP**. It stands for non-deterministic polynomial time complexity class which contains the decision problems that are verifiable by deterministic computations in polynomial time, and solvable in polynomial tine using non-deterministic turning machine (non-deterministic TM). **NP** is a generalisation of **P** class which means that all the problems in **P** are also in **NP**.

  *Example.* An example of problem in **NP** is the integer factorization problem which studies the existence of a factor $f$ such that $1 < f < k$, and $f$ divides $n$ where $n \in \mathbb{Z}$ and $k \in \mathbb{Z}$ [34, 50]. The reason of why the integer factorization problem is not in P-complexity class is that until now, there is no polynomial time algorithm suggested or proved to factorizes a number given in binary of size (m-bit) in time $\mathcal{O}(m^k)$ for some constant $k$.

- **Co-NP**. We say that a decision problem $P_1$ is in Co-NP if the complement of that problem $\overline{P_1}$ is in NP. This implies that the instances that result in no answers can be accepted using a non-deterministic turing machine

in polynomial time. The P-complexity class is a subset of NP and Co-NP complexity classes [27, 30].

- **NP-hard**. It stands for non-deterministic polynomial time hardness. A problem $P1$ is NP-hard if from every problem in NP, there exists a polynomial time reduction to the problem $P1$. So, we say that $P_1$ is **NP-hard** if it is at least as hard as the hardest problem in **NP**.

  *Examples*

  - Optimisation problems such as the optimisation version of the knapsack problem are considered as **NP-hard** problems. For these problems, the optimal solution can not be obtained in polynomial time but approximation solutions with constant approximation ratios can be computed in polynomial time.
  - The halting problem ,which studies the termination of any arbitrary computer program with a given input, is an example which is **NP-hard** but not the **NP-complete** [20].

- **NP-complete**. It refers to the set of decision problems that belong to **NP** and **NP-hard** complexity classes at the same time. It means that a solution can be verified in polynomial time. A decision problem $P_1$ is **NP-complete** if it is in **NP** and there exist a polynomial time reduction from a well-known **NP-complete** problem to the problem $P_1$. The concept of the **NP-completeness** was introduced in 1971 by Cook-Levin theorem [21]. Examples of NP-completer problems are

  - Boolean satisfiability problem (SAT) [21].
  - Knapsack problem [31].
  - Travelling salesman problem [47].
  - Vertex Cover problem [33].
  - Subset sum problem [33].

  For the **NP-complete** problems, the required time to solve them grows rapidly while increasing the problem size $n$. So, researchers usually propose heuristic based methods or approximation algorithms to solve these problems faster (polynomial time approximations are preferred).

- **PTAS**. It stands for polynomial-time approximation scheme. A **PTAS** complexity class always refers to approximation algorithms of the optimisation

problems (**NP-Complete** and **NP-hard** problems). For a given precision value $\epsilon > 0$, the **PTAS** algorithm computes an approximation solution in polynomial time with a ratio of $(1-\epsilon)$ for maximisation problems or $(1+\epsilon)$ for minimisation problems. This time complexity for the **PTAS** algorithms is polynomial in the problem size $n$ in the form of $O(n^c)$ where $c$ may depend on $\epsilon$ [10].

- **FPTAS**. It stands for fully polynomial time approximation scheme. The **FPTAS** class is a subset of the **PTAS** class. A problem $P_1$ is in **FPTAS** if it is in **PTAS** and the algorithm complexity is polynomial in both the size of the problem $n$ and $\frac{1}{\epsilon}$.

  Not all the **NP-hard** problems have **FPTAS** approximations but at least we know from [51] that there exist an **FPTAS** approximation for the knapsack problem.

- **NEXPTIME**. It stands for non-deterministic exponential time. A problem $P_1$ is in **NEXPTIME** if it can be solved by a non-deterministic Turing machine in exponential time in the form of $2^{n^{O(1)}}$.

### 2.2.2 Space Complexity Classes

- PSPACE. It stands for polynomial size. It contains the set of the problems that can be solved using Turing machine using polynomial amount of storage size.

- LOGSPACE. It stands for logarithmic space. It contains the decision problems that are solvable using deterministic Turing machine using logarithmic amount of storage size [19].

### 2.2.3 Reductions

Reduction is a technique that is used to map unknown problem $X$ to its complexity class by reducing the unknown problem $X$ to the form of one of the class well-known problems $Y$. This is known as a reduction from $X$ to $Y$. Hence, we conclude that the problem $X$ can not be harder than the problem $Y$. The mapping procedure from a problem to another is performed using a turing machine computable function. We say that it is a polynomial time reduction if the complexity of the mapping function is $\mathcal{O}(poly(problem\,size))$.

## 2.3 System Dynamics

This section presents the system's dynamics as ordinary differential equation and builds an approximated linearised model which is suitable for our work.

### 2.3.1 Differential Dynamics

Let $V(t)$ be the function describing the temperature in the room at time $t$ and $V(0) = V_0$ be the initial temperature satisfying $V_{min} \leq V_0 \leq V_{max}$. The equation below, taken from [43, 45], describes the change of temperature in a room with one heater:

$$C\frac{dT}{dt} + \lambda V = Q$$

where $C$ is the thermal capacity of the room (kJ/K), $\lambda$ is the thermal conductance between the room and the ambient air (kW/K), and $Q$ is the heat input rate of the heater (kW). If the heater is switched off then $Q = 0$.

Solving this first order differential equation gives us the following formula for $V(t)$.

$$V(t) = \frac{Q}{\lambda} + \left(V_0 - \frac{Q}{\lambda}\right) e^{-\frac{\lambda}{C}t}$$

We can write down this equation as:

$$V(t) = K_1 e^{at} + K_2$$

where $K_1 = V_0 - \frac{Q}{\lambda}$, $K_2 = \frac{Q}{\lambda}$, and $a = -\frac{\lambda}{C}$. So, the temperature inside a room is described by an exponential equation which is hard to analyse. Under some conditions which are satisfied in our system, the exponential equation can be approximated as a linear one. This is shown next.

### 2.3.2 Behaviour Linearisation

Under the natural assumptions that the heater output is much higher than the heat loss and the comfort zone is quite narrow, this exponential behaviour can be approximated well by a linear behaviour. This is because the slope of $V(t)$ at $t = 0$ is $aK_1$ and the most extreme value of the slope of $V(t)$ before the boundary of the comfort zone is reached is $aK_1(1 + (V_{max} - V_{min})/K_1)$.

So, the temperature inside a room heated by a heater can is governed by the following linear equation:

$$V(t) = At + V_0$$

where $A$ represents the slope of the linear equation. The slope $A$ represents the heater heating rate with positive value $A \in \mathbb{Q}_{>0}$ when the heater is operating (ON). On the other hand, when the heater is not functional (OFF), the room will be cooled down due to the difference between temperatures inside and outside the room. Now, the slope $A$ represents the room cooling rate with a negative value $A \in \mathbb{Q}_{<0}$. The main goal is to maintain the temperature inside the comfort zone bounded by $V_{min}$ and $V_{max}$, which are discrete states, while temperature inside the zone is described by a continuous linear equation. Because the linear hybrid system consists of discrete states with linear behaviours between the states, the system can be defined as linear hybrid automata as shown later in Section 2.4.

## 2.4 System Formalisation

### 2.4.1 Preliminaries

Let $\mathbb{0}_N$ and $\mathbb{1}_N$ be $N$-dimensional vectors with all entries equal to 0 and 1, respectively. By $\mathbb{R}_{\geq 0}$ and $\mathbb{Q}_{\geq 0}$ we denote the sets of all non-negative real and rational numbers, respectively. We assume that $0 \cdot \infty = \infty \cdot 0 = 0$. For a vector $v$, let $\|v\|$ be its $\infty$-norm (i.e. the maximum coordinate in $v$). We write $v_1 \leq v_2$ if every coordinate vector of vector $v_1$ is smaller than or equal to the corresponding coordinate in vector $v_2$, and $v_1 < v_2$ if, additionally, $v_1 \neq v_2$ holds.

### 2.4.2 Formal Definition

Motivated by our application of keeping temperature in single/multiple room(s) within comfortable range, we restrict ourselves to safe sets being hyperrectangles, which can be specified by giving its two extreme corner points. A *multi-mode system with discrete costs*, $\mathcal{A}$, henceforth referred to simply as *multi-mode system*, is formally defined as a tuple $\mathcal{A} = (M, N, A, \pi_c, \pi_d, V_{\min}, V_{\max}, V_0)$ where:

- $M$ is a finite set of modes $\{0 \dots |M| - 1\}$.
- $N \geq 1$ is the number of continuous variables in the system. It is also can be seen as the number of dimensions (rooms) where we control their temperatures.
- $A : M \to \mathbb{Q}^N$ is the slope of all the variables in a given mode. It represents the heating/cooling rates of modes. The value $A(m)_{m \in M}$ is positive for heating modes and negative for cooling modes. It may also be zero which means that the temperature remains fixed.

- $\pi_c : M \to \mathbb{Q}_{\geq 0}$ is the cost per time unit spent in a given mode. It represents the running cost while using heaters or air conditioners.
- $\pi_d : M \to \mathbb{Q}_{\geq 0}$ is the cost of switching to a given mode. It can be seen as a set up and maintenance cost.
- $V_{\min}, V_{\max} \in \mathbb{Q}^N$: $V_{\min} < V_{\max}$, define the safe set, $S$, as follows $\{x \in \mathbb{R}^N : V_{\min} \leq x \leq V_{\max}\}$;
- $V_0 \in \mathbb{Q}^N$, such that $V_0 \in S$, defines the initial value of all the variables.

This model will be more simple in the case of only one dimension (room/variable) while using a cost free idle mode to cool the room down when all heaters are being turned off as shown in Chapter 3. Next, we present a simple running example which is useful for understanding the terms being introduced and their definitions. For simplicity, the example addresses the problem of finding safe schedules to keep the temperature inside a single room (one dimension) within a comfort zone. The room is equipped with two heaters only and it is permitted to run any one of them at any time and pay its discrete cost $\pi_d$ at the beginning and continue paying the continuous cost $\pi_c$ as long as you keep it running. It is also allowed to turn the two heaters off to cool the room down without paying any cost.

**Running example 2.** *Suppose we need to keep the temperature inside an office between 18°C and 22°C for $t_{max} = 7$ hours, and the initial temperature inside of it is 18°C. (As we will see later, we can reduce this problem to keeping the temperature inside between $V_{min} = V_0 = 0°C$ and $V_{max} = 4°C$.) We have two heaters, i.e. $|M| = 3$ (considering the idle mode as mode 0), at our disposal: gas (mode 1) and electric (mode 2). Their parameters are $A(1) = 4/3\,[°C/h]$, $A(2) = 2\,[°C/h]$, and $A(0) = -4\,[°C/h]$, i.e. it takes 3 hours for the office to reach the maximum allowable temperature of 22°C when using the gas heater, but just 2 hours using the electric one. It takes 1 hour for the office to cool from 22°C to 18°C, when both of the heaters are off (mode 0). The running costs of the heaters are $\pi_c(1) = 10\,[£/h]$ and $\pi_c(2) = 20\,[£/h]$, and the initial costs of switching each heater are $\pi_d(1) = 30\,[£]$ and $\pi_d(2) = 10\,[£]$. That is, the gas heater is cheaper to run, but more expensive to turn on, e.g. due to a need for regular inspections.* ◁

## 2.5 Schedules and their Cost

In this section, we present definitions for some keywords we use in this thesis. We introduce the term *timed action* which is the most fundamental part in the

definitions of finite/infinite *schedules*. We also demonstrate how the total cost is calculated for a schedule.

**Definition 2.1.** A *timed action* is a pair $(m, t) \in M \times \mathbb{R}_{\geq 0}$ of a mode $m$ and time delay $t > 0$.

**Definition 2.2.** A *schedule* $\sigma$ (of length $k$) with time horizon $t_{\max}$ is a finite sequence of timed actions $\sigma = \langle (m_1,\ t_1),\ (m_2,\ t_2),\ \ldots,\ (m_k,\ t_k) \rangle$, such that $\sum_{i=1}^{k} t_i = t_{\max}$.

**Definition 2.3.** A *schedule* $\sigma$ with infinite time horizon is either an infinite sequence of timed actions $\sigma = \langle (m_1,\ t_1),\ (m_2,\ t_2),\ \ldots,\ (m_k,\ t_k),\ \ldots \rangle$, such that $\sum_{i=1}^{\infty} t_i = \infty$ or a finite sequence of timed actions $\sigma = \langle (m_1, t_1), (m_2, t_2), \ldots, (m_k, t_k) \rangle$, such that $t_k = \infty$.

**Definition 2.4.** The *run* of a finite schedule $\sigma = \langle (m_1, t_1), (m_2, t_2), \ldots, (m_k, t_k) \rangle$ is a sequence of *states* $\text{run}(\sigma) = \langle V_0, V_1, ..., V_k \rangle$ such that, for all $0 \leq i \leq k-1$, we have that $V_{i+1} = V_i + t_i A(m_i)$ as shown in Figure 2.1.

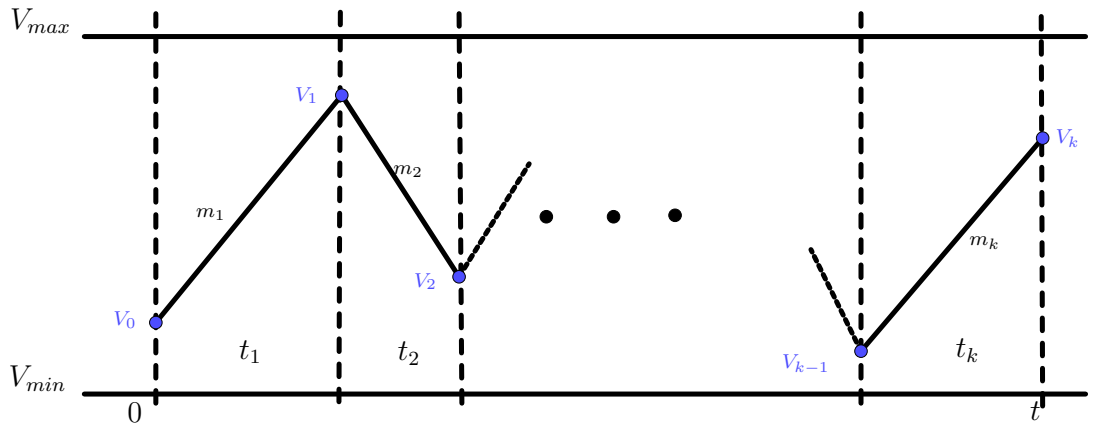



FIGURE 2.1: A finite schedule with its run.

**Definition 2.5.** A finite schedule of length $k$ and its run are called *safe* if $V_{\min} \leq V_i \leq V_{\max}$ holds for all $1 \leq i \leq k$.

**Definition 2.6.** A finite schedule of length $k$ and its run are called $\epsilon$-*safe* if $V_{\min} - \epsilon \cdot \mathbb{1}_N < V_i < V_{\max} + \epsilon \cdot \mathbb{1}_N$ holds for all $1 \leq i \leq k$.

The run of an infinite schedule and its safety and $\epsilon$-safety are defined accordingly. The *total cost* of a schedule $\sigma = \langle (m_1,\ t_1),\ (m_2,\ t_2),\ \ldots,\ (m_k,\ t_k) \rangle$ with a finite time horizon is defined as

$$\pi(\sigma) = \sum_{i=1}^{k} \pi_d(m_i) + \pi_c(m_i) t_i$$

The *limit-average cost* for a finite schedule $\sigma = \langle (m_1, t_1), (m_2, t_2), \ldots, (m_k, t_k)\rangle$ with an infinite time horizon is defined as $\pi_{avg}(\sigma) = \pi_c(m_k)$ and for an infinite schedule $\sigma = \langle (m_1, t_1), (m_2, t_2), \ldots\rangle$ it is defined as

$$\pi_{avg}(\sigma) = \limsup_{k\to\infty} \left( \sum_{i=1}^{k} \pi_d(m_i) + \pi_c(m_i)t_i \right) \Big/ \sum_{i=1}^{k} t_i$$

**Definition 2.7.** A safe finite schedule $\sigma$ that covers a time horizon $t_{max}$ is $\epsilon$-*optimal* if, for all safe finite schedules $\sigma'$ that cover the same time horizon, we have that $\pi(\sigma') \geq \pi(\sigma) - \epsilon$.

**Definition 2.8.** A safe finite schedule is *optimal* if it is 0-optimal.

**Definition 2.9.** A safe infinite schedule $\sigma$ is *optimal* if, for all safe infinite schedules $\sigma'$, we have that $\pi_{avg}(\sigma') \geq \pi_{avg}(\sigma)$.

**Running example 2 continues.** *For instance* $\sigma_1 = \langle (1,3), (0,1), (2,2), (0,1)\rangle$ *and* $\sigma_2 = \langle (1,1), (2,1), (0,\frac{1}{2}), (1,1), (0,\frac{1}{2}), (1,1), (2,1), (0,1)\rangle$ *are both safe finite schedules that last for* 7 *hours as shown in Figure 2.2 and 2.3, respectively. By summing up the contribution of each mode to the overall cost, we get* $\pi(\sigma_1) = (1\cdot 30 + 3\cdot 10) + (1\cdot 10 + 2\cdot 20) = 110\,[£]$ *and* $\pi(\sigma_2) = (3\cdot 30 + 3\cdot 10) + (2\cdot 10 + 2\cdot 20) = 180\,[£]$. *Moreover,* $\sigma_3 = \langle (1,3), (0,1), (2,2), (0,1), (1,3), (0,1), (2,2), (0,1), \ldots\rangle$ *as shown in Figure 2.4 is a safe infinite run with the average cost* $\pi_{avg}(\sigma_3) = 110/7\,[£/h]$. ◁

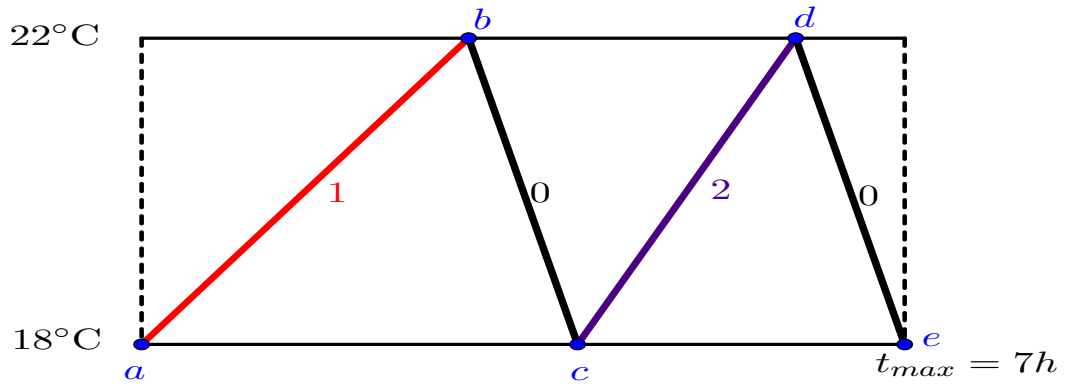



FIGURE 2.2: The finite schedule $\sigma_1$ in example 2.

## 2.6 Conclusions

We showed that, the problem of maintaining the temperature inside a room using HVAC systems can be modelled as a linear hybrid model for narrow comfort

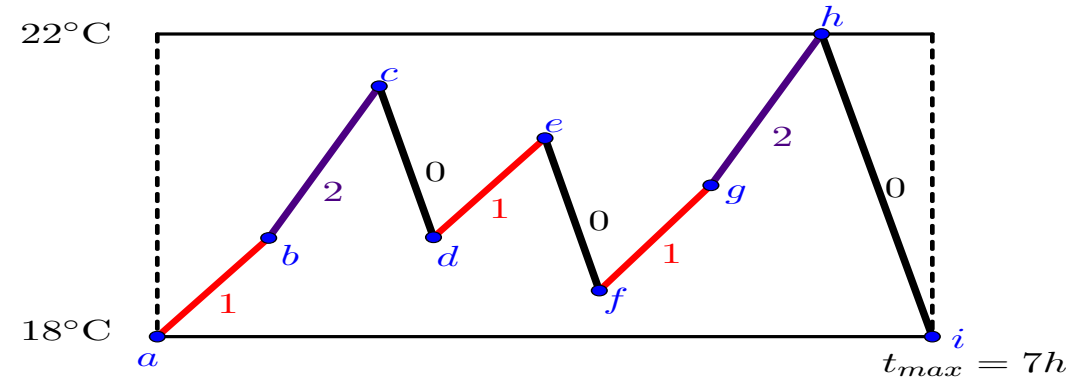



FIGURE 2.3: The finite schedule $\sigma_2$ in example 2.

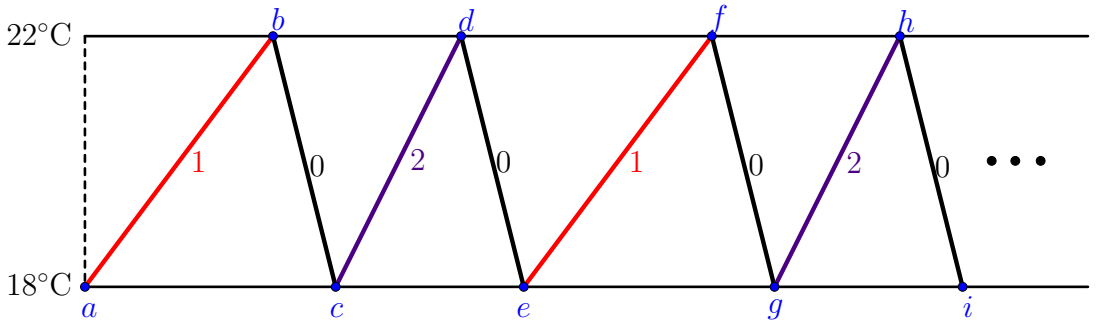



FIGURE 2.4: The infinite schedule $\sigma_3$ in example 2.

zones. Optimal safe schedules are the schedules with the smallest average cost while keeping the temperature of the room inside the comfort zone. This can be done by a better selection of which heater or air-conditioner is turned on or off and exactly when.

# Chapter 3

# Optimisation in a Simple One Dimensional Multi-mode Systems

The simplest subclass of our model is multi-mode systems with a single dimension. It naturally occurs when controlling the temperature in a single room or building to stay in a pleasant range. For this, the system can be in different modes, e.g. the air-conditioning can be switched on or off, or one can choose to switch on an electrical radiator or a gas burner. Each such a configuration can be modelled as a mode of our multi-mode system. Modes have start-up cost (gas burners, e.g., may suffer from some wear and tear when switched on) as well as continuous costs.

In this chapter, we study the temperature control of a single room while paying as small cost as possible as an application of the single dimension system in which we have only heaters without air-conditioners. The heaters can be turned on to increase the room temperature while we turn all the heaters off in order to cool the room down and this mode is known as the *idle* mode.

When keeping an office building in a pleasant temperature range during opening hours, we face a control problem for multi-mode systems with a finite time horizon. The scheduling problem for finite time horizon, which is similar to minimising the total cost incurred during that finite time, is proved in Section 3.3 to be NP-hard while its decision version is NP-complete and significantly more challenging than for the infinite time horizon (LogSpace). The proof is based on a reduction from the unbounded knapsack problem (UKP) which is known to be NP-complete. However, we devise optimal solver using integer programming. We also find approximation solutions with guarantees such as two approximation and FPTAS approximation for the finite time horizon problem. This will be achieved by doing a reduction to the 0-1 Knapsack problem.

## 3.1 Preliminaries

The case we consider here is that we have a multi-mode system with $K$ modes applied in one dimension. For example, this can be used to control the temperature inside a room with $K-1$ heaters. Each heater is denoted by a single mode which can be turned on or off any time. Every time we turn on a heater by selecting its mode, we pay discrete cost only once and continuous cost per unit time. There exist an *idle* mode when all heaters are turned off. We denote the *idle* mode as $m_0$. We refer to the set of non-negative modes by $M^+ = M \backslash \{m_0\}$ (all the modes without the idle mode). The *idle* mode is a cost-free mode and we can use it to cool the room down without paying money. So, the system definition can be simplified as follows.

$$\forall_{m \in M^+} A(m) \geq 0 \wedge \pi_c(m) \geq 0 \wedge \pi_d(m) \geq 0$$

$$A(m_0) < 0 \wedge \pi_c(m_0), \pi_d(m_0) = 0$$

Given a simple linear automaton $\mathcal{A} = (M, A, \pi_c, \pi_d, [V_{min}, V_{max}], V_0, t_{max})$ with $V_{min} > 0$, consider automaton $\mathcal{A}' := (M, A, \pi_c, \pi_d, [0, V_{max} - V_{min}], V_0 - V_{min}, t_{max})$. Note that any finite (infinite) safe schedule $\sigma$ in $\mathcal{A}$ is also safe in $\mathcal{A}'$ and its cost (limit-average cost, respectively) is the same. As a result we have the following observation which allows us to assume $V_{min} = 0$ from now on.

*Observation* 1. Any decision problem regarding ($\epsilon$-)optimal (finite or infinite) schedules for multi-mode systems, can be easily reduced to the same decision problem for multi-mode systems with $V_{min} = 0$.

As we shown in Observation 1 and will be shown in Observation 2, the decision problems for multi-mode systems that we study in this paper can easily be reduced to the same ones for structurally equivalent multi-mode systems with $V_{min} = V_0 = 0$. We can simply argue that by considering a case with $V_{min} < V_0 < V_{max}$. We can use the *idle* mode at the start without paying any cost until the temperature reaches $V_{min}$ or we reach the end of the time period $t_{max}$, whichever comes first.

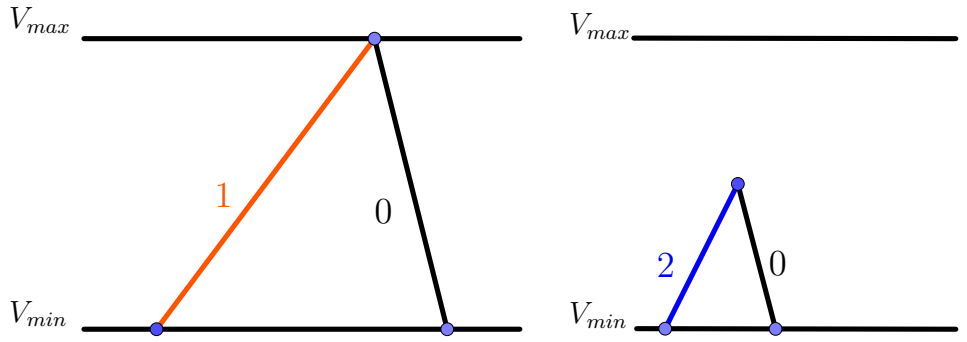
**Definition 3.1.** A *leap* is a sequence of two pairs $(m_k, t_k), (m_{k+1}, t_{k+1})$ in a schedule such that $m_{k+1} = 0$, $A(m_k)t_k \leq V_{max}$, and $A(m_k)t_k + A(m_{k+1})t_{k+1} = 0$. A leap is of *type* $i \in M^+$ iff $m_k = i$.

**Definition 3.2.** A *complete leap* is a leap such that $A(k)t_k = V_{max}$.

Intuitively, a complete leap consists of a phase where the variable is increasing in the same non-idle mode from $V_{min} = 0$ to $V_{max}$, followed by a phase where

the value of the variable is decreasing back to $V_{min} = 0$ in the idle mode. By $\Delta t_i$ and $\Delta\pi_i$ we denote the time duration and the cost of a complete leap of type $i \in M^+$, respectively. Note that $\Delta t_i = V_{max}/A(i) - V_{max}/A(0)$ and $\Delta\pi_i = \pi_d(i) + \pi_c(i) \cdot V_{max}/A(i)$. We also introduce $\pi_e(i) = (\Delta\pi_i - \pi_d(i))/\Delta t_i$ to be the effective continuous cost rate per time unit of using mode $i$ as part of a leap. Note that a leap of type $i$ that lasts for time $t$ has the total cost of $\pi_d(i) + \pi_e(i) \cdot t$.

**Running example 2 continues.** *For instance* $(1,3),(0,1)$ *is a complete leap of type* 1 *and* $(2,1),(0,\frac{1}{2})$ *is an incomplete leap of type* 2. *Their costs are* $(30 + 3 \cdot 10) + (0 + 1 \cdot 0) = 60\,[£]$ *and* $(10 + 1 \cdot 20) + (0 + \frac{1}{2} \cdot 0) = 30\,[£]$, *respectively. We can calculate that* $\Delta t_1 = 4\,[h]$, $\Delta t_2 = 3\,[h]$, $\Delta\pi_1 = 60\,[£]$, *and* $\Delta\pi_2 = 50\,[£]$. *Moreover,* $\pi_e(1) = 7\frac{1}{2}\,[£/h]$ *and* $\pi_e(2) = 13\frac{1}{3}\,[£/h]$. *Note that the cost of this example incomplete leap of type 2 is* $\pi_d(2) + \pi_e(2) \cdot (1 + \frac{1}{2}) = 30\,[£]$, *which matches the cost that we computed earlier.* ◁




(a) A leap of type 1. (b) An incomplete leap of type 2.

FIGURE 3.1: Leaps.

## 3.2 Optimal Schedules

We start with considering the easy case of infinite time horizons, before turning to the interesting case of finite time horizons.

### 3.2.1 Infinite Time Horizon

Let $j = \text{argmin}_{i \in M^+} \Delta\pi_i/\Delta t_i$. Obviously, at all times $t = k \cdot \Delta t_j$ where $k \in \mathbb{N}$, using only complete leaps of type $j$ is the cheapest finite schedule. Consequently, the limit superior of the average cost cannot be smaller than $\Delta\pi_j/\Delta t_j$. At the same time, the simple schedule that only uses complete leaps of type $j$ realises this long-time average. Taking into account that $\text{argmin}_{i \in M^+} \Delta\pi_i/\Delta t_i$ can be computed

using logarithmic space, because multiplication, division and comparison can be [19], we get the following theorem.

**Theorem 3.3.** *An optimal safe infinite schedule can be computed in deterministic* LogSpace.

**Running example 2 continues.** *It is easy to check that* $\sigma_4 = \langle (1,3),(0,1),(1,3),(0,1),\ldots\rangle$ *is an optimal safe infinite run whose long-time average cost is* $\pi_{avg}(\sigma_4) = 15\,[£/h]$. ◁

### 3.2.2 Finite Time Horizon

For finite time horizon, it is more challenging to find the optimal safe schedules. This is because we do not what is the best combination of modes that reduce the cost and cover exactly the time period $t_{max}$. So, in this section we introduce the shape of the optimal schedules for finite time horizon.

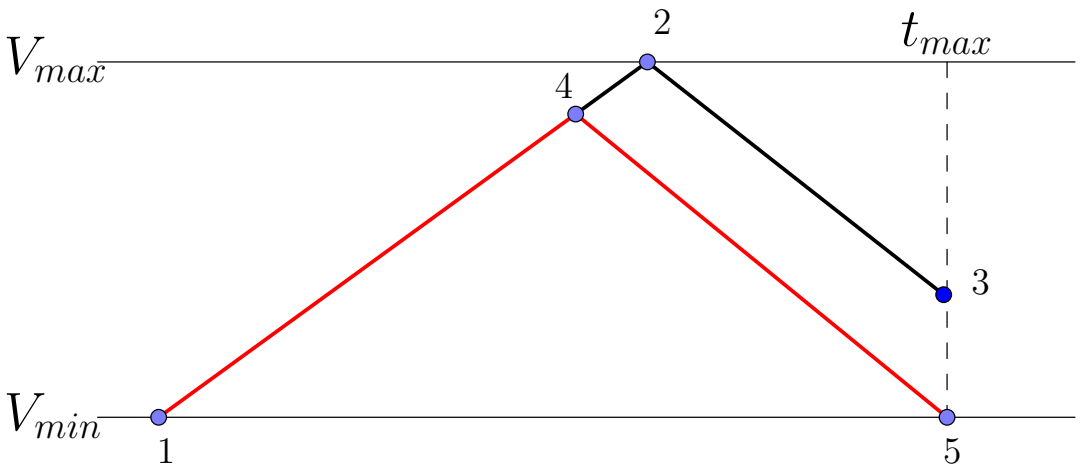



FIGURE 3.2: Any optimal schedule can be assumed to reach value $V_{min} = 0$ at the end. Replacing timed actions $1 \to 2 \to 3$ in a finite safe schedule with timed actions $1 \to 4 \to 5$ reduces the cost of this schedule within its time horizon $t_{max}$.

We start with the following observation.

**Proposition 3.4.** *For every safe schedule* $\sigma$ *there exists a safe schedule* $\sigma'$ *with the same or a lower cost and the value of the variable at the end of* $run(\sigma')$ *equal to* $V_{min} = 0$.

*Proof.* We can see this illustrated in Figure 3.2. Let $t$ be the first point of time during the execution of $\sigma$ that the value of the variable equals $A(0) \cdot (t - t_{max})$. (Note that such a $t \in [0, t_{max}]$ exists.) We then construct $\sigma'$ from $\sigma$ by changing

the behaviour in the last $t_{max} - t$ time units, choosing the idle mode there. As choosing the idle mode incurs no costs, this can only reduce the overall costs. Let we have a schedule $\sigma = \langle (m_1, t_1), \ldots, (m_k, t_k) \rangle$ with a run of $\langle V_0, V_1, \ldots, V_k \rangle$. If $V_k = V_{min}$, it is true. Otherwise, if $V_{min} < V_k \leq V_{max}$, we have two cases.

- If $m_k = m_0$ (i.e. it is the idle mode): We construct the schedule $\sigma' = \langle (m_1, t_1), \ldots, (m_{k-1}, t_{k-1} - (t - t_k)), (m_k, t) \rangle$ that ends with $V_{min}$ and $\pi(\sigma') < \pi(\sigma)$ because the amount of time we use the idle (cost free) mode increased while the time we use the mode $t_{k-1}$ is decreased.

- If $m_k \in M^+$ (i.e. it is not an idle mode): We construct the schedule $\sigma' = \langle (m_1, t_1), \ldots, (m_k, t_k - t), (m_0, t) \rangle$ that ends with $V_{min}$ and $\pi(\sigma') < \pi(\sigma)$ because we added a timed action at the end for the idle mode (cost free) and the time we use the mode $t_k$ is decreased by $t$.

□

In the remainder of this chapter, we assume that all schedules have the property as stated in Proposition 3.4. In fact, we can show that there exists an optimal schedule of a very special form as stated by the following theorem.

**Theorem 3.5.** *Thanks to Proposition 3.4, we can assume that schedules end with $V_{min}$ and hence, for every safe schedule $\sigma$ there exists a safe schedule $\sigma'$ consisting of a sequence of leaps where all but possibly the last one are complete and such that the cost of $\sigma'$ is the same or lower than $\sigma$.*

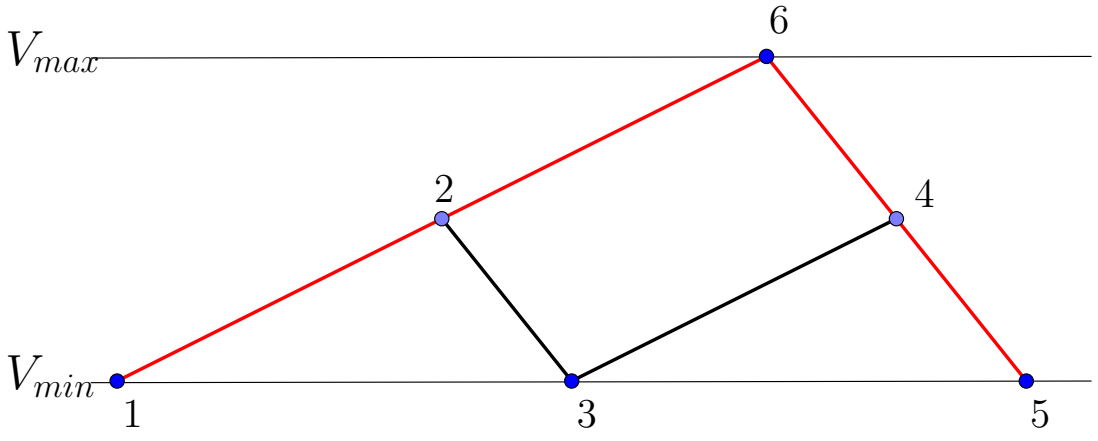



FIGURE 3.3: Two incomplete leaps $1 \to 2 \to 3$ and $3 \to 4 \to 5$ being combined into one leap $1 \to 6 \to 5$.

*Proof.* Let $\sigma = \langle (m_1, t_1), (m_2, t_2), \ldots, (m_k, t_k) \rangle$ be any safe schedule. Define $T_\sigma(m) := \sum_{1 \leq i \leq k : m_i = m} t_i$ to be the total time mode $m \in M^+$ is used for in $\sigma$. We define a

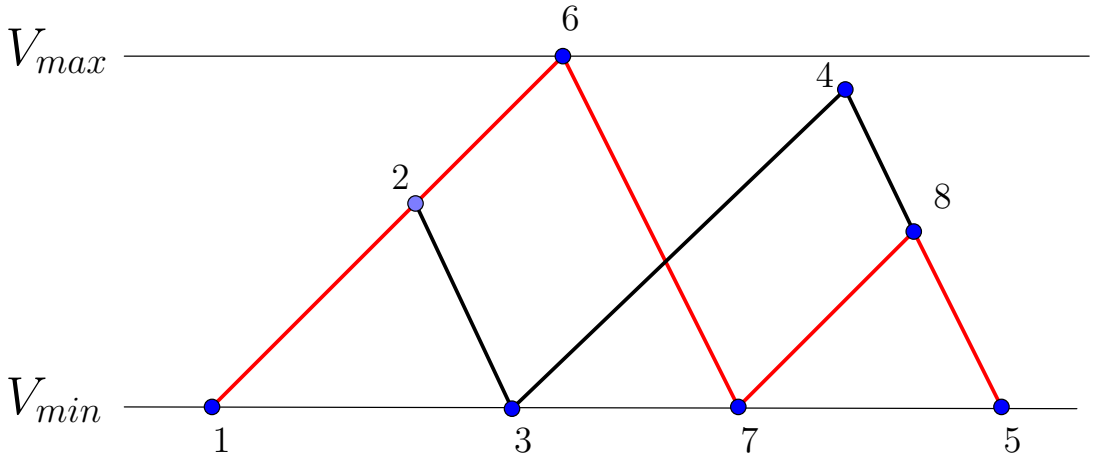



FIGURE 3.4: Two incomplete leaps $1 \to 2 \to 3$ and $3 \to 4 \to 5$ being combined into one complete leap $1 \to 6 \to 7$ and one incomplete one $7 \to 8 \to 5$.

schedule $\sigma''$ as follows: it starts with $\lfloor T_\sigma(m)A(m)/V_{max} \rfloor$ complete leaps of type $m$ for each mode $m \in M^+$. At the end we add for each $m \in M^+$ an incomplete leap starting with a timed action $(m, T_\sigma(m) - V_{max}/A(m)\lfloor T_\sigma(m)A(m)/V_{max} \rfloor)$ if $T_\sigma(m)A(m)/V_{max}$ is not an integer. It is easy to see that $\sigma''$ is safe and no more expensive than $\sigma$, because each mode is used the same amount of time as in $\sigma$ and the number of switches to any mode $m \in M^+$ is the same or smaller. Also, the total time we use the idle mode in $\sigma''$ is the same as the total time we use it in $\sigma$ because it is the only mode that can be used to cool the room down. To construct $\sigma'$ we iterate the following until there is at most one incomplete leap left: take the first two incomplete leaps in $\sigma''$: $(m_1, t_1), (0, t_{01})$ and $(m_2, t_2), (0, t_{02})$. W.l.o.g. the continuous cost for mode $m_1$ is lower, i.e. $\pi_c(m_1) \cdot t_1/(t_1 + t_{01}) \leq \pi_c(m_2) \cdot t_2/(t_2 + t_{02})$. We can then replace these two incomplete leaps by

- $(m_1, (t_1 + t_2 + t_{01} + t_{02}) \cdot t_1/(t_1 + t_{01})), (0, (t_1 + t_2 + t_{01} + t_{02}) \cdot t_{01}/(t_1 + t_{01}))$ if it is a leap, i.e. $A(m_1)(t_1 + t_2 + t_{01} + t_{02}) \cdot t_1/(t_1 + t_{01}) \leq V_{max}$, see Figure 3.3
- one complete leap for $m_1$ and a shorter leap for $m_2$ such that the time delay of the two leaps is $t_1 + t_2 + t_{01} + t_{02}$, see Figure 3.4

This operation cannot increase the cost of the schedule, because the continuous cost of $m_1$ is the same or lower and the number of mode switches is the same or lower. At the same time the number of incomplete leaps is strictly reduced and still the resulting schedule $\sigma'$ ends with $V_{min}$ as shown in Figure 3.3 and 3.4. □

From Theorem 3.5, an optimal schedule exists, because for any fixed time horizon $t_{max}$ there are only finitely many schedules of the form stated and no other schedule can have a better cost than all of them.

When we allow the initial value, $V_0$, of the variable to be non-zero at the beginning, then we can exploit a similar argument to show that it is safe to initially stay in the idle mode until either $t_{max}$ is reached or the value of the variable has fallen to $V_{min} = 0$, whatever happens first.

*Observation* 2. For multi-mode systems with $V_0 > V_{min}$ the following holds. For every safe schedule $\sigma$ there exists a safe schedule $\sigma'$ where initially the idle mode is active until the value of the variable is $V_{min} = 0$ (or, if this is earlier, for the complete duration $t_{max}$), followed by a sequence of leaps, where all but possibly the last one are complete and such that the cost of $\sigma'$ is the same or lower cost than $\sigma$.

## 3.3 NP-Completeness of Finite Time Horizon Optimal Control

In this section, we study the complexity of the optimal control problem for a finite time horizon. As usual, we analyse the complexity of the related decision problem:

> For a given cost $C$, is there a way to control the system in such a way that the total cost incurred for keeping the system in the safe zone for time $t_{max}$ is at most $C$?

We show that this optimal control decision problem is NP-complete. We start by showing its hardness by a reduction from the Unbounded Knapsack problem, which is NP-complete [26]. For this reduction, it suffices to use a simpler problem, where all continuous costs $\pi_c$ are 0. We refer to this problem as 0-cost rate optimal control decision problem.

**Theorem 3.6.** *The* 0*-cost rate optimal control decision problem is NP-hard.*

*Proof.* For the 0-cost rate optimal control problem, cost is only incurred when switching to a non-idle mode. This reduces our continuous optimisation problem to a discrete one, which is easier to relate to the Unbounded Knapsack problem.

In this setting, the natural constraint of the decision problem would be that the time horizon needs to be covered completely, which is reflected by

$$\sum_{i \in M^+} n_i \Delta t_i \geq t_{max}$$

This constraint says that the sum of the time of leaps is at least $t_{max}$. This includes the—possibly incomplete—last leap. Note that, in our discrete setting where the

length of the leap does not influence the cost, the question of whether or not this cycle is complete is irrelevant for the total cost. For this reason, the $n_i$ in this proof refer to all leaps, including the incomplete one.

Under this constraint, we would ask if there are natural numbers $(n_i)_{i\in M^+}$ such that $\sum_{i\in M^+} n_i\pi_d(i) \leq C$. These two constraints together are precisely the constraints used in the Unbounded Knapsack problem, where $C$ represents the volume of the knapsack, $\pi_d(i)$ is the volume of item $i$—such that $\sum_{i\in M^+} n_i\pi_d(i) \leq C$ reflects the constraint volume of the knapsack—$\Delta t_i$ the value of item $i$, and $t_{max}$ the lower bound on the overall value—such that $\sum_{i\in M^+} n_i\Delta t_i \geq t_{max}$ refers to the (decision version of) the optimisation criterion. □

The inclusion in NP of the general cost optimisation decision problem is straightforward, as the problem can be re-written as an integer linear program. Assume that we know the type, $j$, of the incomplete leap at the end of the schedule. We can then solve the decision cost optimisation problem by solving the following integer linear constraint system.

$$\sum_{i\in M^+} n_i\Delta\pi_i + \big(t_{max} - \sum_{i\in M^+} n_i\Delta t_i\big)\pi_e(j) + \pi_d(j) \leq C$$

The first term in this expression is the total cost of the complete leaps and the other one is the total cost of the last (possibly incomplete) leap, whose duration is $t_{max} - \sum_{i\in M^+} n_i\Delta t_i$. Additionally we need the following constraints.

$$\bigwedge_{i\in M^+} n_i \in \mathbb{Z} \quad \wedge \quad \bigwedge_{i\in M^+} n_i \geq 0 \quad \wedge$$

$$t_{max} \geq \sum_{i\in M^+} n_i\Delta t_i \geq t_{max} - \Delta t_j$$

A solution to such a system of integer linear constraints, if it exists, can be guessed and verified in polynomial-time, which shows that the problem is in NP for a fixed $j$. Furthermore, $j$ can be guessed at the same time, which gives us the following theorem.

**Theorem 3.7.** *The finite time horizon optimal control decision problem is NP-complete.*

## 3.4 Optimal Algorithms

### 3.4.1 Integer Linear Programming Algorithm

As we introduced before, the minimisation problem of the total cost being paid in multi-mode single dimension system with discrete costs –demonstrated by the temperature control example using HVAC systems– can be solved using integer linear programming. The formulation of the problem and the algorithm are shown in Algorithm 3. This algorithm produces the optimal solution.

**Algorithm 3** Integer Linear Programming algorithm for the optimal cost problem.

1: Solve the following ILP for all possible $j \in M^+$:

$$\text{Min} \quad \sum_{i \in M^+} n_i \Delta\pi_i + \big(t_{max} - \sum_{i \in M^+} n_i \Delta t_i\big)\pi_e(j) + \pi_d(j)$$

Subject to the following constraints:

$$\bigwedge_{i \in M^+} n_i \in \mathbb{Z} \quad \wedge \quad \bigwedge_{i \in M^+} n_i \geq 0 \quad \wedge$$

$$t_{max} \geq \sum_{i \in M^+} n_i \Delta t_i \geq t_{max} - \Delta t_j$$

2: Pick $j^*$ and the corresponding solution $(n_i)_{i \in M^+}$ with the minimum value of the objective function.

3: **return** schedule consisting of $n_i$ complete leaps of type $i$ for all $i \in M^+$ followed by a leap of type $j^*$ and duration $t_{max} - \sum_{i \in M^+} n_i \Delta t_i$

## 3.5 Constant Factor Approximation Algorithm

We show here an approximation algorithm with a constant relative performance $\leq 2$ for the cost minimisation problem in multi-mode systems which means that this algorithm in the worst case produces approximate solution with 100% absolute error of the optimal solution (twice the optimal solution). We prove that it suffices to pick the cheapest schedule among the ones that only use one of the modes. The complexity of the two approximation algorithm is linear. Building on this constant approximation algorithm, we will show FPTAS for the same problem in the next section.

**Algorithm 4** Constant factor approximation algorithm computing a finite schedule with the total cost at most twice the optimal one.

```
1: MinCost := ∞;  m := 0;
2: for i := 1 to K do
3:     k_i := ⌈t_max/Δt_i⌉;
4:     Cost := π_e(i) · t_max + π_d(i) · k_i
5:     if Cost < MinCost then
6:         MinCost := Cost;  m := i;
7:     end if
8: end for
9: return schedule consisting of ⌊t_max/Δt_m⌋ complete leaps of type m followed by
   at most one more leap of type m for the remaining time t_max − ⌊t_max/Δt_m⌋ · Δt_m
```

Let $k_i := \lceil t_{max}/\Delta t_i \rceil$ denote the minimum number of leaps of type $i$ that have to be used to cover the whole time horizon $t_{max}$ by themselves. Let us introduce the following constant $\alpha := \max\{1, \max_{\{i \in M^+ | k_i \geq 2\}} k_i/(k_i - 1)\}$, where as usual $\max \emptyset = -\infty$. Note that $\alpha \leq 2$, because $k_i/(k_i - 1)$ decreases with $k_i$ and $k_i \geq 2$.

**Theorem 3.8.** *Algorithm 4 runs in deterministic* LogSpace *and returns an $\alpha$-approximate schedule.*

*Proof.* It is straightforward to see that the algorithm can be made to run in deterministic LogSpace, because it suffices to only store $m$ inside the for loop and outputting the value and comparisons between arithmetic expressions can be performed in deterministic LogSpace [19].

To prove that the schedule returned has relative performance at most $\alpha$, we first introduce some useful notation. Let $X_j$ be the value of the *Cost* variable for $i = j$, i.e. $X_j = \pi_e(j) \cdot t_{max} + \pi_d(j) k_j$ for all $j \in M^+$, which is the minimum cost of a schedule that only uses leaps of type $j$. Let us assume w.l.o.g. that the mode picked by Algorithm 4 is 1. Thus, for all $i \in M^+$ we have $X_1 \leq X_i$. Let $\sigma$ be an optimal finite schedule of the form as described in Theorem 3.5. For any $i \in M^+$ let $n_i \in \mathbb{N}$ be the number of complete leaps of type $i$ in $\sigma$. Let the last leap in $\sigma$ be of type $m$ and $0 \leq L \leq \Delta t_m$ be the time that it lasts for. Note that $L = t_{max} - \sum_{i \in M^+} n_i \Delta t_i$. From the definition of $k_i$ we know that $k_i \Delta t_i \geq t_{max} \geq (k_i - 1)\Delta t_i$. It follows that $\Delta t_i / t_{max} \leq 1/(k_i - 1)$ for $k_i \geq 2$. If $n_i \geq 1$ then obviously $\Delta t_i / t_{max} \leq 1$, because otherwise we would have $L < 0$.

Based on these, we note the following estimations:

$$1 - \frac{L}{t_{max}} = \sum_{i \in M^+} \frac{n_i \Delta t_i}{t_{max}} = \sum_{\{i \in M^+ | n_i \geq 1\}} \frac{n_i \Delta t_i}{t_{max}} =$$
$$\sum_{\{i \in M^+ | n_i \geq 1 \& k_i = 1\}} \frac{n_i \Delta t_i}{t_{max}} + \sum_{\{i \in M^+ | n_i \geq 1 \& k_i \geq 2\}} \frac{n_i \Delta t_i}{t_{max}} \leq$$
$$\sum_{\{i \in M^+ | n_i \geq 1 \& k_i = 1\}} \frac{n_i}{k_i} + \sum_{\{i \in M^+ | n_i \geq 1 \& k_i \geq 2\}} \frac{n_i}{k_i} \frac{k_i}{k_i - 1} \leq$$
$$\max\{1, \max_{\{i \in M^+ | k_i \geq 2\}} \frac{k_i}{k_i - 1}\} \sum_{i \in M^+} \frac{n_i}{k_i} = \alpha \sum_{i \in M^+} \frac{n_i}{k_i}$$

Moreover, we have the following. If $k_m \geq 2$ then $\frac{k_m L}{t_{max}} \leq \frac{k_m \Delta t_m}{t_{max}} \leq \frac{k_m}{k_m - 1} \leq \alpha$. If $k_m = 1$ then $\frac{k_m L}{t_{max}} \leq 1 \leq \alpha$, so in fact in both cases $\frac{k_m L}{t_{max}} \leq \alpha$.

We are now ready to give a lower bound on the total cost of the optimal schedule $\sigma$ in terms of $X_1$. The total cost of $\sigma$ is equal to the following expression.

$$\sum_{i \in M^+} (n_i \pi_d(i) + n_i \Delta t_i \pi_e(i)) + \pi_d(m) + L\pi_e(m) =$$
$$\sum_{i \in M^+} \frac{n_i}{k_i} (k_i \pi_d(i) + k_i \Delta t_i \pi_e(i)) + \pi_d(m) + L\pi_e(m) \geq$$
$$\sum_{i \in M^+} \frac{n_i}{k_i} (k_i \pi_d(i) + t_{max} \pi_e(i)) + \pi_d(m) + L\pi_e(m) =$$
$$\sum_{i \in M^+} \frac{n_i}{k_i} X_i + \pi_d(m) + L\pi_e(m) \geq$$
$$\sum_{i \in M^+} \frac{n_i}{k_i} X_1 + \pi_d(m) + L\pi_e(m) \geq$$
$$\frac{X_1}{\alpha} \left(1 - \frac{L}{t_{max}}\right) + \frac{k_m L}{\alpha t_{max}} \pi_d(m) + \frac{L}{\alpha} \pi_e(m) =$$
$$\frac{X_1}{\alpha} \left(1 - \frac{L}{t_{max}}\right) + \frac{L}{\alpha t_{max}} (k_m \pi_d(m) + t_{max} \pi_e(m)) =$$
$$\frac{X_1}{\alpha} \left(1 - \frac{L}{t_{max}}\right) + \frac{L}{\alpha t_{max}} X_m \geq$$
$$\frac{X_1}{\alpha} \left(1 - \frac{L}{t_{max}}\right) + \frac{L}{\alpha t_{max}} X_1 = \frac{X_1}{\alpha}$$

This shows that the cost of $X_1$ is at most $\alpha$ times the optimal cost, which concludes the proof. □

**Running example 2 continues.** *It is easy to check that* $\sigma_5 = \langle (1,3), (0,1), (2,2), (0,1) \rangle$ *is an optimal safe run whose cost is* $\pi(\sigma_5) = 110\,[£]$. *At the same time, a cheapest safe schedule consisting of leaps of type 1 is* $\sigma_6 = \langle (1,3), (0,1), (1, \frac{9}{4}), (0, \frac{3}{4}) \rangle$

*and of type 2 is* $\sigma_7 = \langle(2,2),(0,1),(2,2),(0,1),(2,\frac{2}{3}),(0,\frac{1}{3})\rangle$. *Their costs are* $\pi(\sigma_6) = 2 \cdot 20 + (3 + \frac{9}{4}) \cdot 10 = 112.5$ *and* $\pi(\sigma_7) = 3 \cdot 10 + 4\frac{2}{3} \cdot 20 = 123\frac{1}{3}$. *Hence, Algorithm 4 will return* $\sigma_6$ *and the approximation ratio of this solution is* 1.022. ◁

From the proof of Theorem 3.8 we can easily deduce the following corollary.

**Corollary 3.9.** *Algorithm 4 returns an optimal schedule if* $\pi_d(i) = 0$ *for all* $i \in M^+$.

*Proof.* Analysing the proof of Theorem 3.8 we can make the following observations. If $\pi_d(1) = \pi_d(i) = 0$ then the condition $X_1 \leq X_i$ implies that $\pi_e(1) \leq \pi_e(i)$. The cost of an optimal schedule $\sigma$ is $\sum_{i \in M^+} n_i \Delta t_i \pi_e(i) + L\pi_e(m) \geq \sum_{i \in M^+} n_i \Delta t_i \pi_e(1) + L\pi_e(1) = t_{max}\pi_e(1) = X_1$. □

## 3.6 FPTAS Algorithm

We show that the minimisation problem for multi-mode systems is in FPTAS by a polynomial time reduction to the 0-1 Knapsack problem, for which many FPTAS algorithms are available (see e.g. [32]). The proposed algorithm depends on the dynamic programming solution for the 0-1 knapsack problem. We will therefore introduce the dynamic programming algorithm for solving the 0-1 knapsack problem in the next sub-section and in Section 3.6.2, we devise an FPTAS solution using a reduction from the unbounded knapsack problem to the 0-1 knapsack problem.

### 3.6.1 Dynamic Programming for 0-1 Knapsack

The dynamic programming solution for the 0-1 knapsack problem searches for the minimum knapsack weight that yields the exact profit $v$ [32]. The problem can be defined as finding $o(n, v)$ which is the minimal knapsack weight that yields value $v$ using subset of items $\{1, .., n\}$ where $v_n$ represents the item cost and $w_n$ is the item weight. We can define $o(n, v)$ recursively as follows:

$$o(n,v) = \begin{cases} min\{o(n-1,v), w_n + o(n-1, v-v_n)\} & \text{if } v_n \leq v \textbf{ and } n \geq 1 \\ 0 & \textbf{otherwise if } v \leq 0 \\ +\infty & \textbf{otherwise} \end{cases}$$

Our optimisation problem is similar but with a minimisation objective. We therefore modify the dynamic programming algorithm so that we search for the

maximum time span that can be covered when paying no more than cost c. As shown in Algorithm 5, the program is being executed over one vector $T$ where $T(c)$ represents the maximum time that can be covered by paying at most a total cost of $c$. The solution produced by this algorithm is affected by rounding errors. The rounding error occurs as the knapsack problem now is considered over rational numbers (not integers any more) which was shown in [53] to be strong NP-complete problem. So, the rounding errors can not be removed while solving the knapsack problem with rational values (weights and profits).

**Algorithm 5** Dynamic Programming algorithm for the optimal cost problem *DP(*$M, \Delta\pi_i, \Delta t_i, t_{max}, c_{max}$*)*. where $i \leq |M|$, and $c_{max}$ is the maximum possible cost where $c \leq c_{max}$

```
1: initialize T(k) = 0 where k ∈ {1..⌈c_max⌉};
2: while SOMETHING CHANGES IN T do
3:     for k = MaxIndex downto 1 do
4:         for h = 1 To n do
5:             T(k) = max(T(k), T(k − Δπ_h) + Δt_h);
6:         end for
7:     end for
8: end while
9: Return the minimum cost index c = k where T(k) ≥ t_max.
```

### 3.6.2 FPTAS Approximation Algorithm

Let $c^*$ be the $\alpha$-approximation, which can be computed using Algorithm 4, of the optimal cost $o^*$. Since $\alpha \leq 2$, to get an approximation to our optimal cost problem with a relative performance $\rho$, it suffices to find a solution with $c^*\rho/2$ absolute performance. We split this into two equal parts of $\epsilon = c^*\rho/4$. An optimal solution to the knapsack instance that we produce will provide us with a schedule with cost no greater than $\epsilon$ over the optimal one. Moreover, a solution to the knapsack instance with $\delta$ absolute error will provide a schedule with an $\epsilon + \delta$ absolute error. Therefore, it suffices to set $\delta = \epsilon$ to find a schedule with $\rho$ relative performance. In our reduction, the value of the resulting knapsack problem is at most $4|M|$ times the optimal cost for safe schedules, so by using $\rho' = \rho/(8|M|)$, for the resulting knapsack problem, we will find a near optimal solution with a relative performance $\rho$ for multi-mode systems. The running time of this procedures is in $\mathcal{O}(\mathsf{poly}(1/\rho)\mathsf{poly}(|M|)\mathsf{poly}(\text{size of the knapsack instance}))$. This suffices to establish the inclusion of the cost minimisation problem for multi-mode systems in FPTAS.

Let $\sigma$ be an optimal safe schedule consisting of a sequence of leaps where all but possibly the last one are complete. Such a sequence exists due to Theorem 3.5. Let $m^* \in M^+$ be the mode used in the last leap in $\sigma$. Note that we can try all modes as candidates for $m^*$.

As shown in Algorithm 6, we build the following items for this knapsack problem instance for each mode $m \in M^+$: $\{(2^i \cdot \Delta t_m, 2^i \cdot \Delta\pi_m) \mid i \in \mathbb{N} \wedge 2^i \cdot \Delta\pi_m \leq c^* \wedge 2^i \cdot \Delta t_m \leq t_{max}\}$. Let $i^* \in \mathbb{N}$ be the smallest natural number such that $2^{-i^*} \cdot (\Delta\pi_{m^*} - \pi_d(m^*)) \leq \epsilon$. For $m^*$ we add the following extra multiset of items: $\{(2^{-i} \cdot \Delta t_{m^*}, 2^{-i} \cdot (\Delta\pi_{m^*} - \pi_d(m^*))) \mid i \in \mathbb{Z}_+ \wedge i \leq i^* \wedge 2^{-i} \cdot (\Delta\pi_{m^*} - \pi_d(m^*)) \leq c^*\}$ and additionally $(2^{-i^*} \cdot \Delta t_{m^*}, 2^{-i^*} \cdot (\Delta\pi_{m^*} - \pi_d(m^*)))$, which is a copy of an element already in the multiset. Let $t_\Sigma$ be the time span of all items in this knapsack instance. We set the volume of this 0-1 knapsack instance to be $t_\Sigma - t_{max}$.

The just produced knapsack problem has the following properties.

- The size of its description is polynomial in the size of the original problem, including the relative performance
- If there is an incomplete leap of $m^*$ in $\sigma$, it can be overestimated by stringing together the fractional copies of leaps (without start-up cost), so that we do not exceed the volume by $2^{-i^*} \cdot \Delta t_{m^*}$ or more, and if there is no incomplete leap in $\sigma$, one complete leap of $m^*$ of $\sigma$ can be replaced by all of these fractional copies of leaps of $m^*$. The remaining complete leaps can be replaced by sums of complete leaps of the respective type.
- The volume of these items is $\geq t_{max}$. Let $v^*$ be the value of these items. Then $v^* + \pi_d(m^*) - \epsilon \leq o^* \leq v^* + \pi_d(m^*)$.
- Let $V_\Sigma$ be the value of all items in the multiset. For any solution to the knapsack problem with value $V$ we get a schedule $\sigma'$ with cost $\leq V_\Sigma - V + \pi_d(m^*)$.

**Lemma 3.10.** *Solving this knapsack instance with a relative performance of $\rho/(8|M|)$ gives us a safe schedule with relative performance of $\rho$.*

**Corollary 3.11.** *Solving the optimal control problem for multi-mode systems with relative performance $\rho$ takes*

$$\mathcal{O}(\mathsf{poly}(1/\rho)\mathsf{poly}(\textit{size of the instance}))$$

*time and is therefore in FPTAS.*

**Algorithm 6** FPTAS approximation algorithm for the optimal cost problem *FPTAS(*$M, \forall_{i\leq|M|}\Delta\pi_i, \forall_{i\leq|M|}\Delta t_i, t_{max}, \epsilon, o^*$*)* where $o^*$ is the optimal estimation cost.

1: Create the array MinCost$(1 \ldots |M^+|)$ and initialise all the items to $\infty$.

2: For each mode $m \in M^+$ we build the following items for this knapsack problem instance: $\{(2^i{\cdot}\Delta t_m, 2^i{\cdot}\Delta\pi_m) \mid i \in \mathbb{N} \wedge 2^i{\cdot}\Delta\pi_m \leq c^* \wedge 2^i{\cdot}\Delta t_m \leq t_{max}\}$.

3: For every $m^* \in M^+$ which is the last incomplete leap mode do steps **from** 4 **to** 8.

4: Find the smallest $i^* \in \mathbb{N}$ such that $2^{-i^*} \cdot (\Delta\pi_{m^*} - \pi_d(m^*)) \leq \epsilon$.

5: For $m^*$ we add the following extra multiset of items: $\{(2^{-i}{\cdot}\Delta t_{m^*}, 2^{-i}{\cdot}(\Delta\pi_{m^*} - \pi_d(m^*))) \mid i \in \mathbb{Z}_+ \wedge i \leq i^* \wedge 2^{-i} \cdot (\Delta\pi_{m^*} - \pi_d(m^*)) \leq c^*\}$ and additionally $(2^{-i^*} \cdot \Delta t_{m^*}, 2^{-i^*} \cdot (\Delta\pi_{m^*} - \pi_d(m^*)))$, which is a copy of an element already in the multiset.

6: Scale all the items' costs using a scaling factor $K$, where $\Delta\pi_m = \lceil \Delta\pi_m / K \rceil, \forall_{m\in M^+}$ where $K = \epsilon \times (max_{\forall_{m\in M^+}}\Delta\pi_m)/|M^+|$ .

7: Run the DP Algorithm 5 for the 0-1 knapsack problem over the new instances with horizontal cost line limit of $(\lceil o^*/K_2 \rceil)$ and store the result in *MinCost(j)* where j is the index of $m^*$.

8: After obtaining the solution, do inverse scalling.

$$MinCost(j) = MinCost(j) \cdot K$$

9: Find the minimum cost over all the results stored in the array *MinCost*.

---

**Running example 2 continues.** For an FPTAS approximation with $\epsilon = 5\%$, the reduction procedure into the 0-1 knapsack problem is shown in Figure 3.5. The complete leap of type 1 shown in Figure 3.5-a has a time duration $\Delta t_1 = 4\,[\text{h}]$. So, as shown in Figure 3.5-c we cannot fit two copies of the complete leaps into the time horizon bounded by 7 [h]. The complete leap of type 2 shown in Figure 3.5-b has a time duration $\Delta t_2 = 3\,[\text{h}]$. So, as shown in Figure 3.5-e we can fit only two copies of the complete leaps $(a \rightarrow b \rightarrow c \rightarrow d \rightarrow e)$ into the time horizon bounded by $7\,[\text{h}]$. This extra item has a time duration of $6\,[h]$ and a cost of $100\,[\pounds]$. The optimal solution as we calculated before is $110\,[\pounds]$ and for an FPTAS with a precision ratio 5% precision, the precision value is $\epsilon = 5.5\,[\pounds]$. The fractions of the complete leap (partial leaps) of type 1 are shown in 3.5-d. The smallest $i^* \in \mathbb{N}$ such that $2^{-i^*} \cdot (\Delta\pi_1 - \pi_d(1)) \leq \epsilon$ is 3 which means that the set of fractions are $(1 \rightarrow 2''' \rightarrow 3''')$, $(1 \rightarrow 2'' \rightarrow 3'')$ and $(1 \rightarrow 2' \rightarrow 3')$ with values

$\{(2\,[h], 15\,[£]), (1\,[h], 7.5\,[£]), (\frac{1}{2}\,[h], 3.75\,[£])\}$, respectively where every element in this set is a tuple $(\Delta t, \Delta\pi)$. The fractions of the complete leap of type 2 is shown in 3.5-f. The smallest $i^* \in \mathbb{N}$ such that $2^{-i^*} \cdot (\Delta\pi_2 - \pi_d(2)) \leq \epsilon$ is 3 which means that the set of fractions are $(1 \to 2''' \to 3''')$, $(1 \to 2'' \to 3'')$ and $(1 \to 2' \to 3')$ with values $\{(\frac{3}{2}\,[h], 20\,[£]), (\frac{3}{4}\,[h], 10\,[£]), (\frac{3}{8}\,[h], 5\,[£])\}$ where every element in this set is a tuple $(\Delta t, \Delta\pi)$. So, the total number of elements in the reduced 0-1 knapsack problem are 9 items. ◁

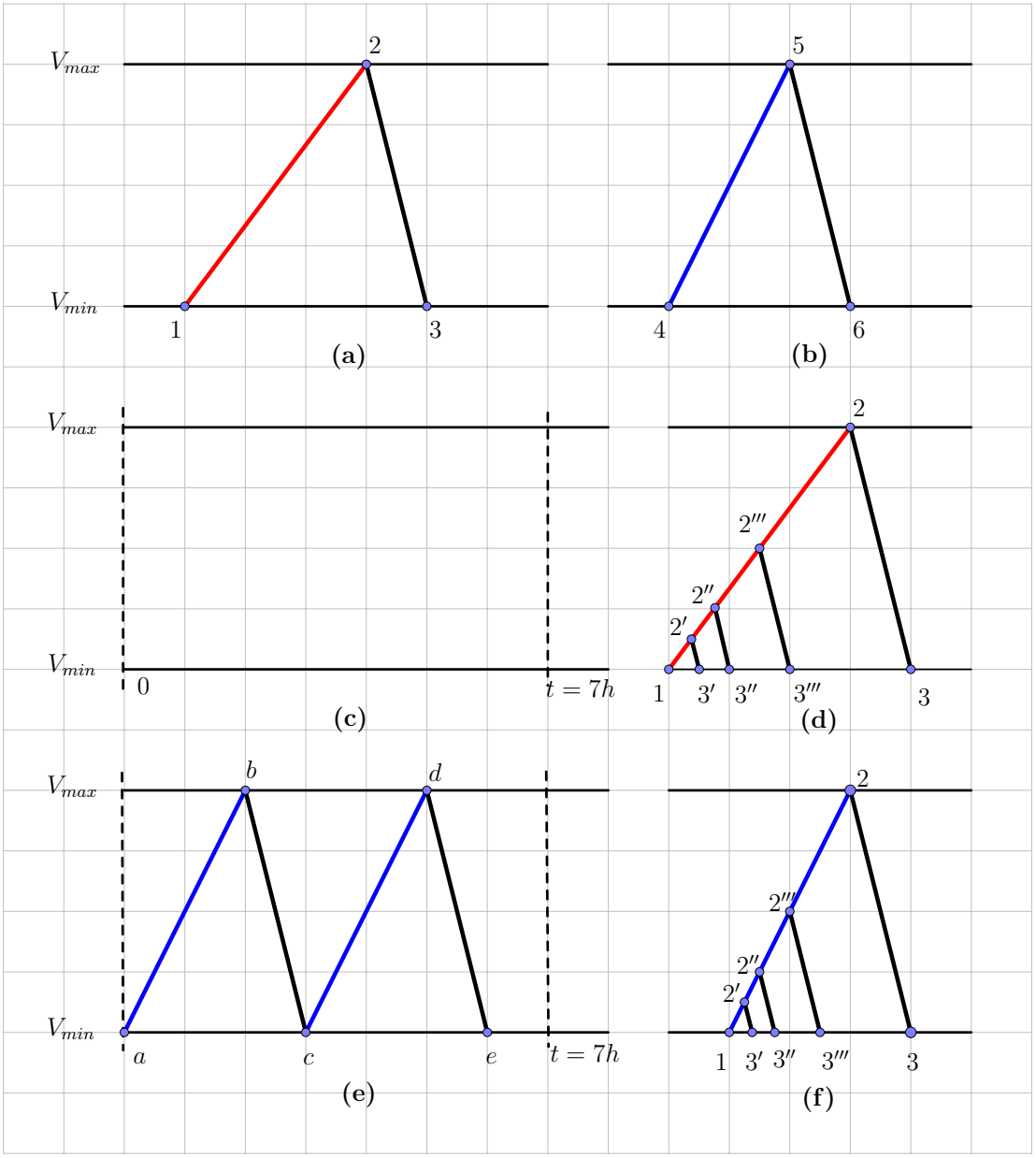



FIGURE 3.5: An example for the reduction into 0-1 knapsack that results in an FPTAS approximation with $\epsilon = 5\%$. The complete leaps of types 1 and 2 are shown in (a) and (b), respectively.The multiples and fractions of the complete leap of type 1 is shown in (c) and (d), respectively. The multiples and fractions of the complete leap of type 2 is shown in (e) and (f), respectively.

**Running example 2 continues.** The problem now can be considered as solving a knapsack problem two times and pick the best answer. The two problems are when we consider only the original items and all the multiples with the fractions only from type 1 while the the other one with fractions only from type 2. After running the FPTAS algorithm shown in Algorithm 6 over the instances shown in Figure 3.5 we get the schedule that contains the complete leap which is shown by $(1 \to 2 \to 3)$ from Figure 3.5-a followed by another complete leap from Figure 3.5-b shown by $(4 \to 5 \to 6)$. The cost of this schedule is $60 + 50 = 110\,[£]$ which is the same cost as the optimal schedule. Note that we add a discrete cost $\pi_d(1) = 30$ one time for all the fractional leaps we use from type 1 and similarly for fractional leaps of type 2. ◁

## 3.7 Conclusions

Linear hybrid systems are computationally challenging. In particular, safety and reachability are undecidable already for three variables. In this chapter, we have identified the class of simple multi-mode systems as a class that arises naturally when studying the optimal control of heating or cooling systems with only heaters and the *idle* model can be used any time to cool the room down: there is only one continuous variable (the temperature in our setting) in addition to the time. Although it was to be expected that the optimal control for this model is decidable, the fact that this problem is both NP-complete and admits an FPTAS was not. Only a small number of NP-hard problems admit an FPTAS, i.e. can be approximated with relative precision $\rho$, in polynomial time in the size of the input and $1/\rho$. Most NP-hard problems can be shown to be inapproximable within a constant relative performance in polynomial time unless P=NP. The existence of FPTAS, besides offering a cheap approximation in every desired precision, often indicates that good standard solvers will normally behave well.

Summing up, we have identified a simple subclass of linear hybrid automata with an easy (LogSpace) optimal control problem over an infinite time horizon, and an optimal control problem over a finite time horizon, which is fast to approximate (FPTAS). We believe that this class is of interest because, broadly speaking, it is just tractable enough. Adding to the collection of classes with de-facto efficient algorithms it expands the set of problems that we can handle. As the next step, the model where there can be multiple modes with negative slopes (i.e. $A(i) < 0$) apart from the idle mode which is analysed in Chapter 4. Such a generalisation, however, breaks down the existence of an optimal schedule consisting of leaps, which was crucial for the development of an FPTAS algorithm for this problem.

# Chapter 4

# Optimisation in General One Dimensional Multi-mode Systems without an idle mode

## 4.1 Introduction

In this chapter, we still study the optimisation problem in multi-mode one dimensional systems. We consider the general case where it is allowed to have modes with negative slope (i.e, $A(m) < 0$) instead of having only one such mode (which is the cost-free *idle* mode used in Chapter 3). Considering the motivating example, we allow using heaters to heat the room up and air-conditioners to cool it down. We first begin by showing an example where the optimal schedule form we presented in Chapter 3 is not working anymore. We overcome this problem by showing how the optimal schedule looks like. We present a cost non-increasing operations that transform any safe schedule to the optimal shape. Based on this, we propose FPTAS approximation as well as a 3-approximation algorithm that runs in $\mathcal{O}(|\mathcal{A}|^7)$ time. We prove that the optimisation problem still NP-hard and its decision version is NP-complete.

### 4.1.1 Motivation Example

Suppose we need to keep the temperature inside an office between 18°C and 22°C for finite time horizon $t_{max} = 9$ hours. Assume the system has three modes with the specifications shown in Table 4.1. Note that we do not have a cost-free idle mode any more and this is the main reason of introducing this motivating example.

The optimal schedule shape presented in Chapter 3 consists of complete leaps and possibly with the last one incomplete. In our example, a leap consists of

| | $\pi_d$ | $\pi_c$ | $\mathrm{A}(m_i)$ |
|---|---|---|---|
| $m_1$ | 1 [£] | 40 [£/h] | 0.5 [°C/$h$] |
| $m_2$ | 1 [£] | 20 [£/h] | 2 [°C/$h$] |
| $m_0$ | 100 [£] | 20 [£/h] | $-4$ [°C/$h$] |

TABLE 4.1: An example for the multi-mode one-dimensional system general case.

a timed action that uses either modes $m_1$ or $m_2$ to heat the room up followed by another timed action that uses modes $m_0$ to cool it down. So, if we try to generate schedules that follow the same form of the optimal schedules introduced in Chapter 3, we may have schedules as follows

- Schedule $\sigma_1 = \langle(1, 8), (0, 1)\rangle$ shown by $(a \to f \to g)$ in Figure 4.1 uses a complete leap of modes $m_1$ followed by $m_0$. The cost of this schedule is $\pi(\sigma_1) = 1 + 40 \times 8 + 100 + 20 \times 1 = 441$£.
- Schedule $\sigma_2 = \langle(2, 2), (0, 1), (2, 2), (0, 1), (2, 2), (0, 1)\rangle$ shown by $(a \to b \to c \to d \to e \to f \to g)$ in Figure 4.1 uses complete leaps of modes $m_2$ followed by $m_1$ with average cost of $\pi(\sigma_2) = (1 + 20 \times 2 + 100 + 20 \times 1) \times 3 = 483$£.
- Any other schedule $\sigma_3$ that contains leaps of modes $m_2$ followed by $m_0$ and incomplete leap of mode $m_1$ followed by mode $m_0$ has an average cost of $\pi(\sigma_3) > \pi(\sigma_2)$.

But, as shown in Figure 4.2 we may have another schedule that has a different shape from what was presented in Chapter 3 and it has lower cost. The schedule $\sigma_4 = \langle(1, 4), (2, 1), (0, 1), (1, 3)\rangle$ shown by $(a \to b \to c \to d \to e)$ covers the same time period $t_{max}$ with an average cost of $\pi(\sigma_4) = 1 + 40 \times 4 + 1 + 20 \times 1 + 100 + 20 \times 1 + 1 + 40 \times 3 = 423$£ which is smaller than the cost of all the schedules presented before. This schedule has features.

- Because we pay cost not only for heating the room but also for cooling it, the schedule does not have to end at $V_{min}$.
- The switching point between the modes can be anywhere between $V_{min}$ and $V_{max}$.

This example makes the optimal schedule form shown in the simplifying case in Chapter 3 not suitable for the general case and we have to find the form of optimal

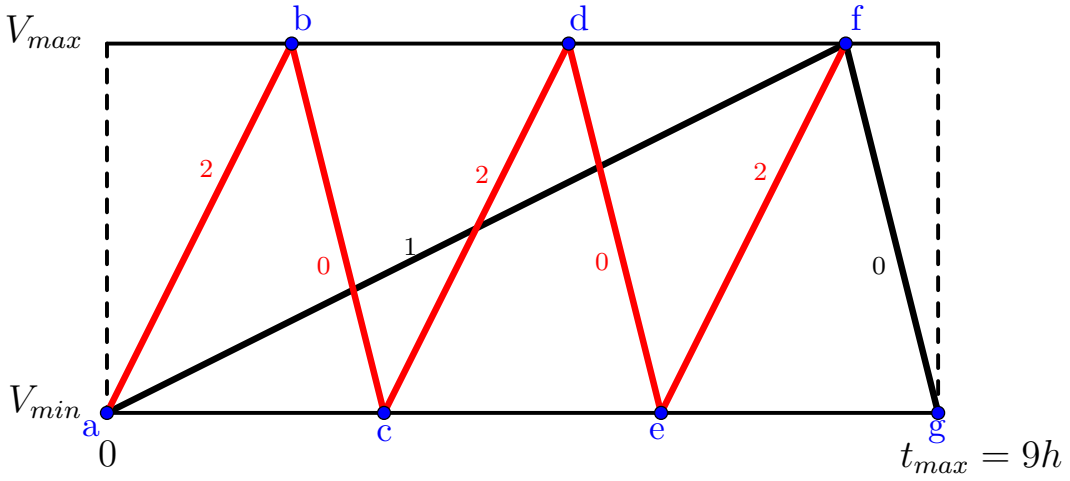



FIGURE 4.1: Schedules that follow the form of the optimal schedule introduced in Chapter 3.

schedules that fits the general case of the optimisation in multi-mode 1-dimensional systems.

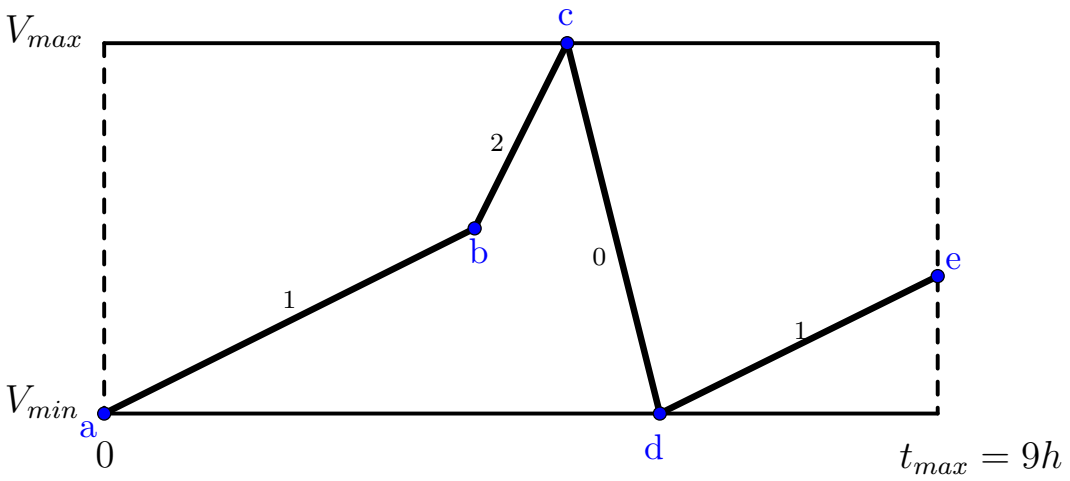



FIGURE 4.2: Example of a schedule that violates the form of the optimal schedule introduced in Chapter 3.

## 4.2 Preliminaries

Let $M^+ = \{m \mid A(m) > 0\}$ and $M^- = \{m \mid A(m) < 0\}$. Recall that $M^0 = \{m \mid A(m) = 0\}$. We will call a mode an *up mode, down mode, or zero-mode* if $m \in M^+$, $m \in M^-$, or $m \in M^0$, respectively. Similarly, the *trend* of a timed action $(m, t)$ is *up, down, flat* if $m$ is an up, down, zero-mode, respectively. For any subsequence of timed actions $\sigma' = \langle (m_i,\ t_i),\ \ldots,\ (m_j,\ t_j) \rangle$ in a schedule $\sigma$, whose run is $run(\sigma) = \langle V_0, V_1, \ldots, V_k \rangle$, we say that $\sigma'$ *starts at state* $v$ and *ends at*

*state* $v'$ iff $v = V_{i-1}$ and $v' = V_j$. We use the same terminology for a single timed action (in this case this subsequence has length 1).

## 4.3 Structure of Finite Control in One-dimension

We show in this section that any finite schedule in one-dimension can be transformed without increasing its cost into a schedule, which follows one of finitely many regular patterns. We redefine the "leap" component we introduced in Chapter 3 as follows.

**Definition 4.1.** A *partial leap* is a pair of consecutive timed actions $(m_i, t_i), (m_{i+1}, t_{i+1})$ in a schedule such that $m_i \in M^+$, $m_{i+1} \in M^-$, and $A(m_i)t_i + A(m_{i+1})t_{i+1} = 0$. A partial leap is *complete* if $A(m_i)t_i = V_{\max} - V_{\min}$. We will simply refer to complete leaps as *leaps*.

There are $|M^+ \times M^-|$ types of leaps. A leap is of *type* $(m, m') \in M^+ \times M^-$ iff $m_i = m$ and $m_{i+1} = m'$. Let $\Delta t_m$ and $\Delta \pi_m$ denote the time and cost it takes for an up mode $m$ to get from $V_{\min}$ to $V_{\max}$ or a down mode $m$ to get from $V_{\max}$ to $V_{\min}$. Note that $\Delta t_m = |(V_{\max} - V_{\min})/A(m)|$ and $\Delta \pi_m = \pi_d(m) + \pi_c(m) \cdot \Delta t_m$. By $\Delta t_{m,m'}$ and $\Delta \pi_{m,m'}$ we denote the time duration and the cost of a leap of type $(m, m') \in M^+ \times M^-$, respectively. Note that $\Delta t_{m,m'} = \Delta t_m + \Delta t_{m'}$ and $\Delta \pi_{m,m'} = \Delta \pi_m + \Delta \pi_{m'}$.

Any schedule $\sigma$ can be decomposed into three sections that we will call its *head, leaps, and tail*. The *head section* ends after the first timed action that ends at $V_{\min}$. The *leaps section* contains only leaps of possibly different types following the head section. Finally, the *tail section* starts after the last leap in the leaps section has finished. Note that any of these sections can be empty and the tail section can in principle contain further leaps. We show here that, for any schedule of length at least three, there exists another one with the same or a smaller cost, whose head and tail sections follow one of the 10 patterns presented in Section 4.3.2 (Figure 4.7 and Figure 4.8, respectively), where *partial up/down* means that the next state is not at the border. For each of these patterns, there exists an example which shows that an optimal schedule may need to use such a pattern and hence it is necessary to consider it. In order to prove this, we first need to define several cost-nonincreasing and safety-preserving operations that can be applied to schedules. These will later be applied in Theorem 4.9 to transform any schedule into one of the just mentioned regular patterns. The formal definition of these operations are shown as well as figures explain the idea behind them in an easy way.

### 4.3.1 Operations

Let $\sigma$ be any safe finite schedule. We show that while looking for an optimal finite schedule of the multi-mode one dimensional systems, we can restrict our attention to angular schedules only. The concept of angular schedule is defined as follows.

**Definition 4.2.** We call a finite schedule $\sigma$ *angular* if there are no two consecutive timed actions $(m_i, t_i), (m_{i+1}, t_{i+1})$ in $\sigma$ such that $A(m_i) = A(m_{i+1})$.

We assume that all finite schedules are angular. If we have a schedule that has more than one timed action with zero modes, we can just shift all timed actions with the zero-modes to the beginning of the schedule and select only one zero-mode out of them –the mode with the minimal cost per unit time– and use it for all the times.

**Proposition 4.3.** *For every finite safe schedule with time horizon $t_{max}$ there exists a safe schedule with the same or lower cost, in which at most one zero-mode is used at the very beginning.*

*Proof.* Let $\sigma$ be a finite safe schedule with timed actions $(m_1, t_1), (m_2, t_2), \ldots, (m_l, t_l)$ that use zero-modes (i.e. $m_i \in M^0$ for all $i \leq l$) for some timed actions inside the schedule $\sigma$. If no such timed actions exist then $\sigma$ is already in the form requested and we are done. Let $m_0 = \text{argmin}_{i \leq l} \pi_c(m_i)$ be the zero-mode among the ones used by $\sigma$ with the lowest continuous cost. We construct a new safe schedule $\sigma'$ by first removing from $\sigma$ all timed actions that use a zero-mode. We then add at the very beginning a single timed action $(m_0, \sum_{i \leq l} t_i)$. It is easy to see that such defined $\sigma'$ is safe and its total cost is equal or lower than that of $\sigma$. □

Henceforth, we assume that all finite schedules use at most one zero-mode timed action and only at the very beginning.

Following that, we can assume that $\sigma$ is angular and only contains at most one timed action with a zero-mode, and if it contains one, this action occurs at the very beginning. Unless explicitly stated, the operations below are defined for timed actions with up or down trend only.

#### *Rearrange Operation*

The first operation that we need is the *rearrange* operation, which simply changes the order of any subsequence of timed actions with the same trend. Figure 4.3 shows the *rearrange* operation applied to three timed actions 1-2-3 with modes $m_1, m_2, m_3$ results in 1'-2'-3' with modes $m_2, m_3, m_1$. The *rearrange* operation is a cost non-increasing operation as we use the same modes for the same amount of time.

**Definition 4.4** (Rearrange Operation)**.** Let $(m_i, t_i), \ldots, (m_j, t_j)$ be any subsequence of $\sigma$ such that either $\forall_{i \leq l \leq j}\, m_l \in M^-$ or $\forall_{i \leq l \leq j}\, m_l \in M^+$ hold. Note that any permutation of the timed actions $(m_i, t_i), \ldots, (m_j, t_j)$ will result in a new schedule $\sigma'$ which is safe and has the same total cost as $\sigma$.

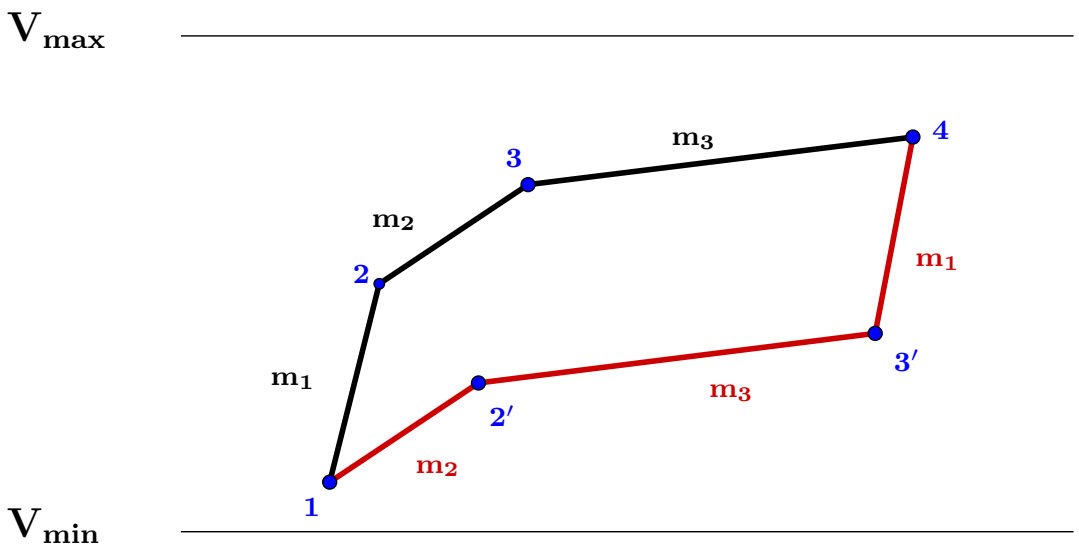



FIGURE 4.3: Rearrange operation.

### *Shift Operation*

The *shift* operation cuts any subsequence of timed actions that start and end at the same state, $V$, and pastes this subsequence after any timed action that ends at $V$. The effect of the *s*hift operation can be seen in Figure 4.4 where the partial leap 1-2-3 which will be moved after the (complete) leap 3-4-5.

**Definition 4.5** (Shift Operation)**.** Let the run of our finite schedule $\sigma = \langle (m_1, t_1), (m_2, t_2), \ldots, (m_k, t_k) \rangle$ be $\langle V_0, V_1, \ldots, V_k \rangle$. For any $i \leq j \leq l$ such that $V_i = V_l = V_j$ holds, we can move the whole subsequence of timed actions $(m_i, t_i), \ldots, (m_{j-1}, t_{j-1})$ just after $(m_{l-1}, t_{l-1})$ in $\sigma$ to obtain a new safe schedule with the same cost. Specifically, the new schedule will look as follows: $\langle (m_1, t_1), \ldots, (m_{i-1}, t_{i-1}), (m_j, t_j), \ldots, (m_{l-1}, t_{l-1}), (m_i, t_i), \ldots, (m_{j-1}, t_{j-1}), (m_l, t_l), \ldots, (m_k, t_k) \rangle$ Analogously, in the same situation, we can also move the whole subsequence of timed actions $(m_j, t_j), \ldots, (m_{l-1}, t_{l-1})$ just after $(m_{i-1}, t_{i-1})$ in $\sigma$ to obtain a new safe schedule with the same cost.

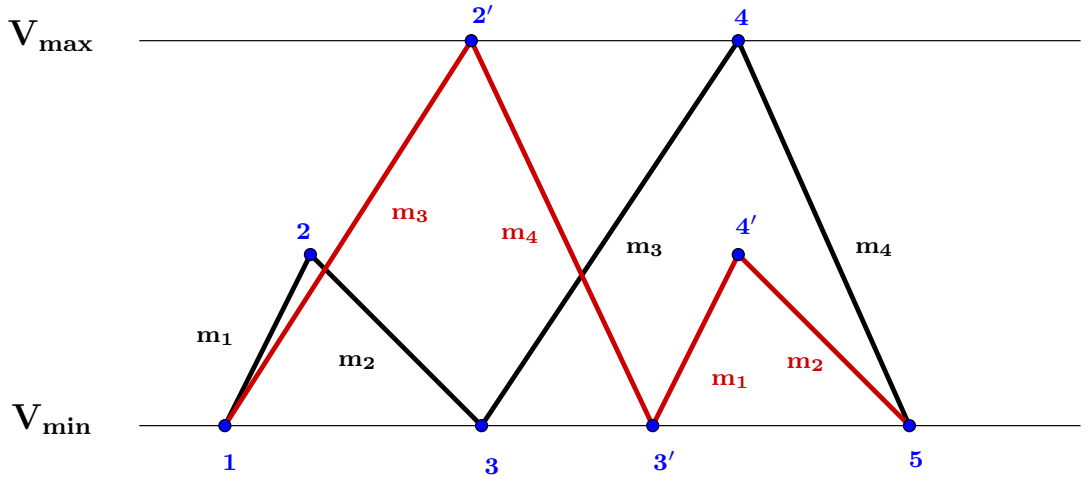



FIGURE 4.4: Shift operation.

### *Shift-down Operation*

The *shift-down* operation can rearrange any subsequence of timed actions that start and end at the same state and move them after any timed action that ends at $V_{\min}$. We can see an example of applying this operation in Figure 4.5. The shift-down operation is applied to timed actions $m_{i+1}, m_{i+2}$. These actions are rearranged to move after point 5, which becomes point 3' (i.e. following timed action $m_{i+3}$).

**Definition 4.6** (Shift-Down Operation)**.** Let the run of our finite schedule $\sigma = \langle (m_1, t_1), (m_2, t_2), \ldots, (m_k, t_k) \rangle$ be $\langle V_0, V_1, ..., V_k \rangle$. For any $i \leq j$ and $l$ such that $V_i = V_{j+1} = V_{\max}$ and $V_{l+1} = V_{\min}$, we can "rotate" the whole subsequence of timed actions $(m_i, t_i), \ldots, (m_j, t_j)$ and move it just after $(m_l, t_l)$ in $\sigma$ to obtain a new safe schedule $\sigma'$ with the same cost. Specifically, let $d = \operatorname{argmin}_{i \leq b < j} V_{b+1}$. Note that if we rotate the subsequence of actions in the way to start with timed action $(m_d, t_d)$ then we will never encounter a lower state than the start state, because $d$ was the lowest point along this subsequence of timed actions. Specifically, the new schedule $\sigma'$ will look as follows $\langle (m_1, t_1), \ldots, (m_{i-1}, t_{i-1}), (m_{j+1}, t_{j+1}), \ldots, (m_l, t_l), (m_d, t_d), \ldots, (m_j, t_j), (m_i, t_i), \ldots, (m_{d-1}, t_{d-1}), (m_{l+1}, t_{l+1}), \ldots, (m_k, t_k) \rangle$.

### *Wedge Operation*

The most complicated operation we define is the *wedge* operation. It acts on three consecutive timed actions in a schedule and simultaneously shrinks the middle action while extending the other two, or stretches the middle action while shrinking the other two. We can see its behaviour in Figure 4.6. Intuitively, it moves the timed action $m_2$ parallelly up or down, until either the timed action $m_1$ is removed

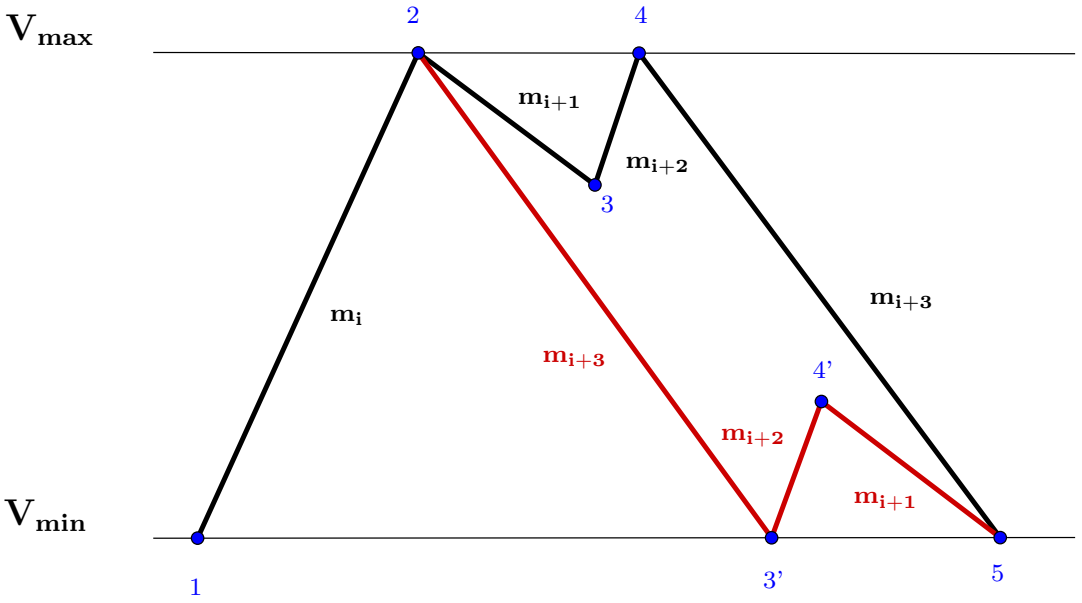



FIGURE 4.5: Shift-down operation.

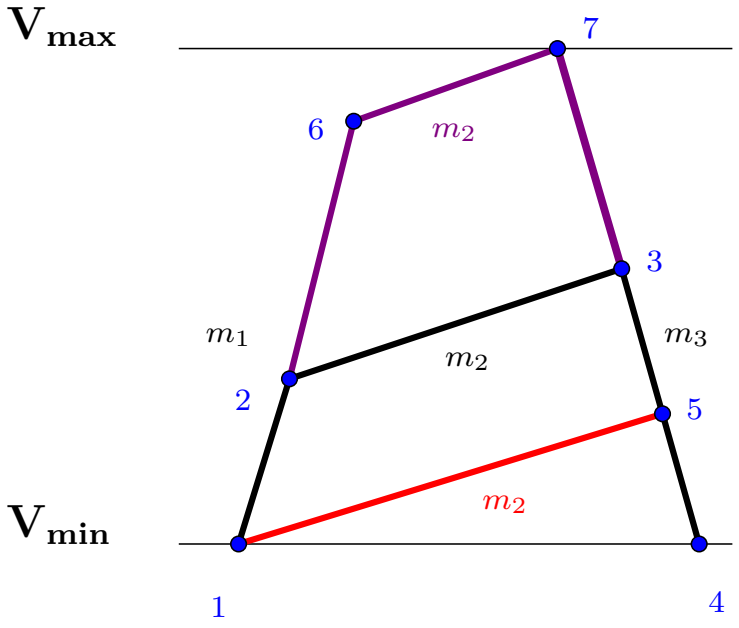



FIGURE 4.6: *Wedge* operation.

or $m_2$ ends at $V_{\max}$. Figure 4.6 shows an example of applying the *Wedge* operation to three timed actions $m_1, m_2, m_3$. This operation is a (parallel) translation of the action $m_2$, which changes the time duration of each of theses actions. After this operation either the $m_2$ line touches $V_{\min}$, which would remove $m_1$ from the schedule, or the $m_2$ line touches $V_{\max}$, which would change a state along the run of the schedule to be at the border.

As we show later, the direction depends on the cost gradient, but as the cost delta function of this operation is linear, one of these directions is cost-nonincreasing.

**Definition 4.7** (Wedge Operation)**.** Let $\sigma = \langle (m_1, t_1), (m_2, t_2), \ldots, (m_k, t_k) \rangle$ be a finite safe schedule whose run is $\langle V_0,\ V_1,\ \ldots,\ V_k \rangle$. Let $\tau = \langle (m_i,\ t_i),\ (m_{i+1},\ t_{i+1}),\ \ (m_{i+2},\ \ t_{i+2}) \rangle$ be any three consecutive timed actions in which exactly two consecutive timed actions have the same trend. It suffices to consider the case where $A(i) > A(i+1) > 0$ and $A(i+2) < 0$ as all other cases are very similar. Notice that if $A(i+1) > A(i)$ then we can simply change the order of $(m_i, t_i), (m_{i+1}, t_{i+1})$ using the rearrange operation defined earlier. Furthermore, we only define this operation in the case where $V_{i-1} = V_{i+2}$. This is the only situation we need this operation for and it is easy to generalise this further.

Let $\alpha \in [0, \ldots, \alpha_{max}]$ where $\alpha$ represents the amount of time subtracted from the time we use mode $m_{i+1}$. The value of $\alpha = 0$ when the time $t_i = 0$ for the mode $m_i$ (i.e. the starting point of the timed action $(m_{i+1}, t_{i+1})$ is $V_{min}$) while $\alpha = \alpha_{max}$ when the ending point of the timed action $(m_{i+1}, t_{i+1})$ is $V_{max}$. So, for any $\alpha \in [0, \ldots, \alpha_{max}]$, consider the sequence of timed actions

$$\tau' = \langle (m_i, \alpha \frac{A(i+2)}{A(i+2) - A(i)}), (m_{i+1}, t - \alpha), (m_{i+2}, t_i + t_{i+1} + t_{i+2} - (t+\alpha) - \alpha \frac{A(i+2)}{A(i+2) - A(i)}) \rangle$$

Let us replace $\tau$ by $\tau'$ in $\sigma$ to get $\sigma'$ whose run is $\langle V'_0,\ \ V'_1,\ \ \ldots,\ \ V'_k \rangle$. We claim that $V_{i-1} = V'_{i-1} = V_{i+2} = V'_{i+2}$, so the runs of $\sigma$ and $\sigma'$ can only differ at their $i$-th and $i+1$-th states. At the same time notice that $\pi_c(\sigma') - \pi_c(\sigma)$ is a linear function of $\alpha$ as a sum of linear functions. As a result its minimum is attained at the smallest or largest permissible value of $\alpha$. Moreover, the value of $t$ can be calculated by studying the sequence $1 \to 5 \to 4$ in Figure 4.6 where $t = (A(i+2)(t_i + t_{i+1} + t_{i+2})/(A(i+2) - A(i+1))$. Similarly, the value of $\alpha_{max}$ can be calculated by studying the sequence $1 \to 6 \to 7 \to 4$ in Figure 4.6 where

$$\alpha_{max} = \frac{V_{max}(A(i+2) - A(i))/A(i+2) - A(i)(t + t_i + t_{i+1} + t_{i+2}) - tA(i+1)}{-A(i+1) - A(i)}$$

***Resize Operation***

Finally, we define the *resize* operation that will be used the most in our procedure. The resize operation requires one parameter $t \in \mathbb{R}$ and can act on any two consecutive timed actions in a schedule. Intuitively, if $t < 0$, this operation decreases the total time of this pair of timed actions by $|t|$ while changing only the middle state between these two timed actions along the run of the schedule. If $t > 0$, this operation increases the duration of this pair of timed actions by $t$ while again changing only the state between them along the run. If $t > 0$ then we will also refer to this operation as the *stretch* operation and if $t < 0$ as the *shrink* operation with parameter $-t > 0$. If the stretch and shrink operations are simultaneously applied with the same parameter $t$ to two non-overlapping pairs of timed actions, the result is a schedule with the same time horizon as before, but with a possibly different total cost. We will call a *flexi* any subsequence of length 2 in a schedule such that both shrink and stretch operations can be applied to it for some $t > 0$. The non-overlapping pairs of timed actions are flexis where each flexi contains different timed actions which means that the intersection between the two flexis can not be a timed action but it can be a common point. A simultaneous application of these two operations to flexis is demonstrated in Figure 4.9 and 4.10. Figure 4.9 shows the shrink and stretch operations being applied to two up-up flexis. The 1-2-3 one is stretched by $t$, which results in 1-4-5, and 1'-2'-3' is shrunk by $t$, which results in 4'-5'-3'. Note that 3 and 5 (also, 1' and 4') are the same states but shifted in time. In fact, all states along the run of the schedule stay the same apart from 2 and 2', and as a result the schedule stays safe. Also, Figure 4.9 presents the shrink and stretch operations being applied to two up-down flexis.

**Definition 4.8** (Resize Operation)**.** Let $\sigma = \langle (m_1,\ t_1),\ \ldots,\ (m_k,\ t_k)\rangle$ whose run is $\langle V_0,\ V_1,\ \ldots,\ V_k\rangle$. For $i < k$ and $t \in \mathbb{R}$, let resize$(\sigma, i, t)$ be a schedule $\sigma'$ identical to $\sigma$ apart from timed actions $(m_i, t_i), (m_{i+1}, t_{i+1})$ being replaced by $(m_i, t'_i), (m_{i+1}, t'_{i+1})$ in the following way, where we distinguish among several cases. If $t > 0$ then we will also refer to this operation as the *stretch* operation and if $t < 0$ as the *shrink* operation.

**(up-up)** If $0 < A(m_i) < A(m_{i+1})$ then let $t'_i = t_i + \beta t + t$ and $t'_{i+1} = t_{i+1} - \beta t$ where

$$\beta = \frac{A(m_i)}{A(m_{i+1}) - A(m_i)} \geq 0$$

Let resize-domain$(\sigma, i) := [-t_i/(\beta + 1), t_{i+1}/\beta]$. Note that $\pi_c(\sigma') - \pi_c(\sigma) = ((\beta + 1)\pi_c(m_i) - \beta\pi_c(m_{i+1})) \cdot t$.

If $0 < A(m_{i+1}) < A(m_i)$ then let $t'_i = t_i - \beta \cdot t$ and $t'_{i+1} = t_{i+1} + \beta \cdot t + t$ where

$$\beta = \frac{A(m_{i+1})}{A(m_i) - A(m_{i+1})} \geq 0$$

Let resize-domain$(\sigma, i) := [-t_{i+1}/(\beta+1), t_i/\beta]$. Note that $\pi_c(\sigma') - \pi_c(\sigma) = ((\beta+1)\pi_c(m_{i+1}) - \beta\pi_c(m_i)) \cdot t$.

**(up-down)** Here $0 < A(m_i)$ and $A(m_{i+1}) < 0$ holds. Let $t'_i = t_i + \beta t$ and $t'_{i+1} = t_{i+1} - \beta t + t$ where

$$\beta = \frac{-A(m_{i+1})}{A(m_i) - A(m_{i+1})} \geq 0$$

Let resize-domain$(\sigma, i) := [-\min\{t_i/\beta, t_{i+1}/(1-\beta)\}, (V_{\max} - V_i)/(\beta A(m_i))]$. Note that $\pi_c(\sigma') - \pi_c(\sigma) = (\beta\pi_c(m_i) + (1-\beta)\pi_c(m_{i+1})) \cdot t$.

**(down-up)** Analogous to **up-down** case.

**(down-down)** Analogous to **up-up** case.

**(flat)** If $(m_1, t_1)$ is a zero-mode action in $\sigma$, then let resize$(\sigma, 0, t)$ be equal to $\sigma$ where the first action is replaced by $(m_1, t_1 + t)$. Let resize-domain$(\sigma, 0) := [-t_1, t_{\max} - t_1]$ and notice that $\pi_c(\sigma') - \pi_c(\sigma) = \pi_c(m_1) \cdot t$

**(last-action)** If $(m_k, t_k)$ is the last action in $\sigma$, then let resize$(\sigma, k, t)$ be equal to $\sigma$ where the last action is replaced by $(m_k, t_k + t)$.
Let resize-domain$(\sigma, k) := [-t_k, \max\{(V_{\max} - V)/A(m_k), (V_k - V_{\min})/A(m_k)\}]$ and notice that $\pi_c(\sigma') - \pi_c(\sigma) = -\pi_c(m_k) \cdot t$

Consider two non-overlapping flexis at positions $i$ and $j$ in a safe schedule $\sigma$. Let $\sigma' = \text{resize}(\sigma, i, t)$ be the resulting schedule of applying the resize operation with parameter $t$ to the $i$-th and $i+1$-th timed actions in $\sigma$ and resize-domain$(\sigma, i)$ be the maximal closed interval from which $t$ can be picked to ensure that $\sigma'$ is safe. Similarly, let $\sigma'' = \text{resize}(\sigma, j, -t)$ and $\sigma''' = \text{resize}(\text{resize}(\sigma, i, t), j, -t))$. Note that $\sigma'''$ has the same time horizon as $\sigma$ and is safe as long as $t \in$ resize-domain$(\sigma, i) \cap$ resize-domain$(\sigma, j)$ and let us denote this closed interval by $I$. Furthermore, $\pi(\sigma''') - \pi(\sigma) = \pi(\sigma') - \pi(\sigma) + \pi(\sigma'') - \pi(\sigma)$ because the two flexis did not overlap. As it is shown in the definition before, both $\pi(\sigma') - \pi(\sigma)$ and $\pi(\sigma'') - \pi(\sigma)$ are linear functions in $t$ in the interior of $I$. As a result, $\pi(\sigma''') - \pi(\sigma)$ is also a linear function in $t$ and so its minimum value is achieved at one of the endpoints of $I$. Also, at such an endpoint, one of the time actions in these two flexis will disappear and as a result the total cost would be reduced even further. It follows, that there is an endpoint of $I$ such that selecting it as $t$ will not increase the cost of the

schedule, but it will remove a flexi from $\sigma$. As the zero-mode timed action and the last timed action in a schedule can have flexible time delay, we can also define the resize operation for them in a similar way. As a result, we can apply the resize operation with parameter $t$ to any of these (including a flexi) and with parameter $-t$ to the other. Reasoning as above, there is a value for $t$ such that the cost of the resulting schedule does not increase, the schedule remains safe, and at least one of the timed actions is removed from $\sigma$ or one more state along the run of $\sigma$ becomes $V_{\min}$ or $V_{\max}$.

### 4.3.2 Transforming Schedules into Optimal Ones

**Theorem 4.9.** *For every safe schedule $\sigma$ in a one-dimensional multi-mode system there exists a safe schedule $\sigma'$ whose head section matches one of the patterns in Figure 4.7, tail section matches one of the patterns in Figure 4.8, and $\pi(\sigma') \leq \pi(\sigma)$ holds. Furthermore, it suffices to consider only 44 combinations of these head and tail patterns, and the length of all of them is at most five.*

*Proof.* We will repeatedly apply combination of shrink and stretch operations to flexis until we remove all non-overlapping ones. Note that after each such an application either a timed action is removed or one more state along the run of $\sigma$ becomes equal to $V_{\max}$ or $V_{\min}$.

We claim that the following steps will transform $\sigma$ to a suitable $\sigma'$:

1. as long as there are at least one pair of non-overlapping flexis then shrink one and stretch the other until a timed action is removed or a new state at the border is created;

2. once there is only one flexi left or two overlapping ones, use the shift or shift-down operation to move them to the end of the schedule;

3. if the first timed action is flat, pair it with the remaining flexi to remove one of them using the shrink-stretch operation combination;

4. if the last state of $run(\sigma)$ is not at the border and a flexi or flat timed action remains after the previous step, they should be paired with each other for the shrink-stretch operation combination;

5. if two overlapping flexis exist, use the wedge operation to resolve them;

6. finally, if the tail section still does not follow any of the patterns, apply the shift-down operation to the (unique) segment that starts and ends at $V_{\max}$.

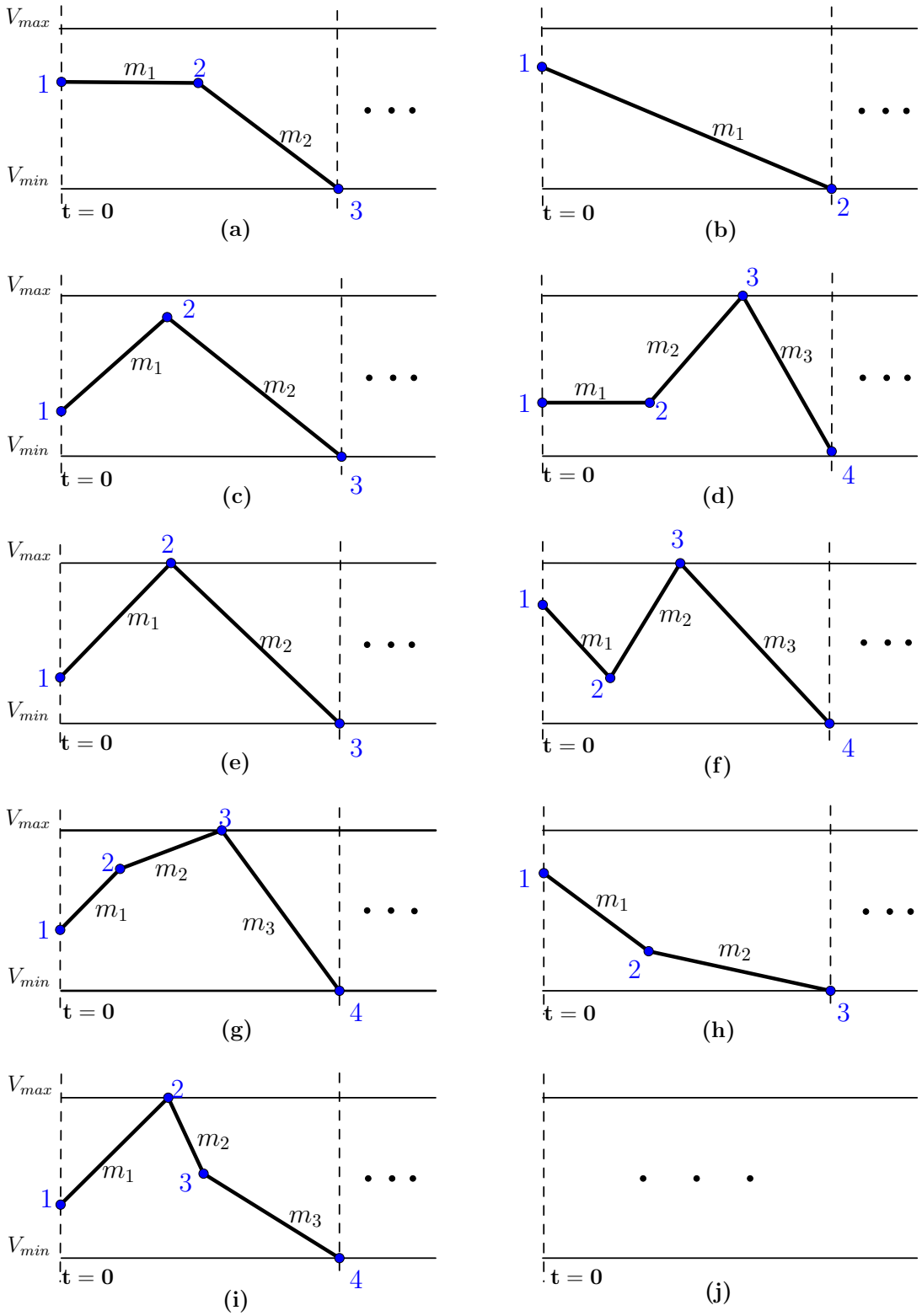



FIGURE 4.7: Ten possible head patterns: (a) flat+down (b) down (c) partial-up+down (d) flat+up+down (e) up+down (f) partial-down+up+down (g) partial-up+up+down (h) partial-down+down (i) up+partial-down+down and (j) empty.

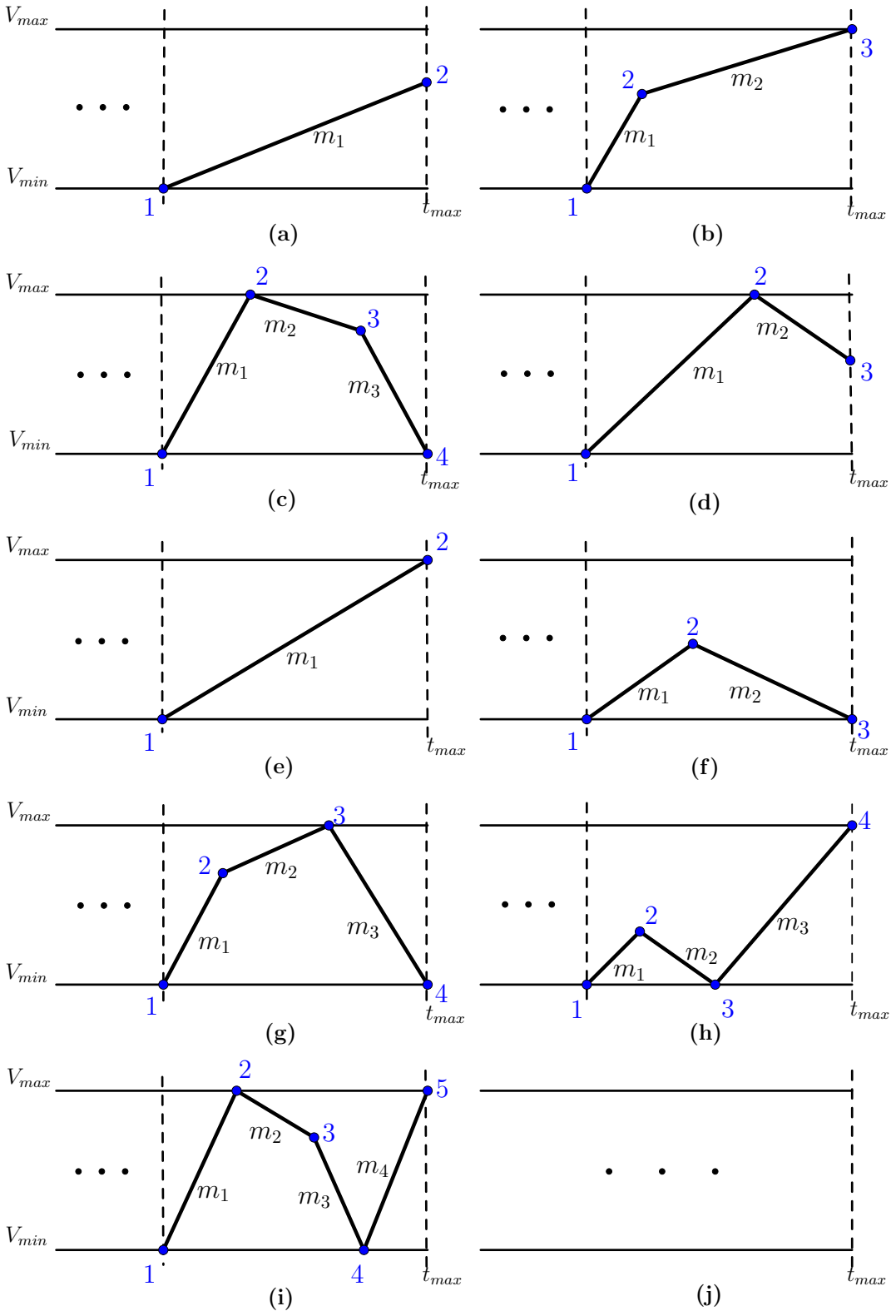



FIGURE 4.8: Ten possible tail patterns: (a) partial-up (b) partial-up+up (c) up+partial-down+down (d) up+partial-down (e) up (f) partial-up+down (g) partial-up+up+down (h) partial-up+down+up (i) up+partial-down+down+up and (j) empty.

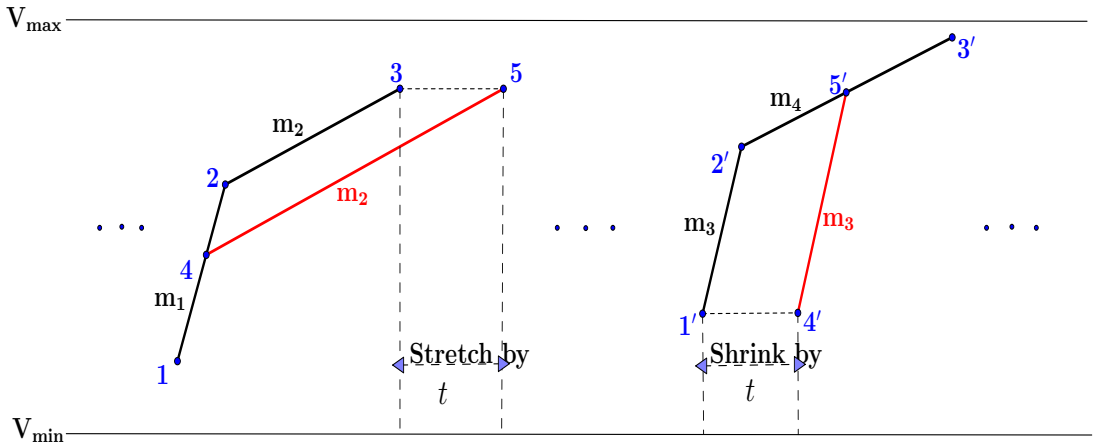



FIGURE 4.9: Shrink and stretch operations being applied to two up-up flexis.

A graphical representation of this procedure when applied to an example schedule can be seen in Example 4.1. It is easy to see that the first step of this procedure will stop eventually because $\sigma$ has a finite number of timed actions and states along its run. The rest of the steps of this procedure just try to reduce the number of possibilities for the head and tail sections. Note that, apart from the initial state, there can be only one state, along the run of the resulting $\sigma'$, which is not at the border. This is because otherwise a shrink-stretch or wedge operation could still be applied. Drawing all possible patterns with one point not at the border and eliminating the ones that are inter-reducible using one of these operations, results in Figure 4.7 for the head section and Figure 4.8 for the tail section.

If we try to combine all these head and tail pattern together then this would result in $10 \cdot 10 = 100$ possible combinations. However, as just mentioned, there can be only one point not at the border or a zero-mode timed action in a schedule so these combinations of head and tail patterns can be reduced further. In particular, any head pattern can be combined with tail patterns (e) and (j), but only (b), (e), (j) head patterns can be combined with the remaining tail ones. Therefore, there are $10 \cdot 2 + 3 \cdot 8 = 44$ combined patterns and it is easy to check that none of them has length larger than five (important for Theorem 4.13).

Note that the schedule's optimal form contains at most one point of flexibility between $V_{min}$ and $V_{max}$. □

**Example 4.1.** *Suppose that a safe-schedule is given in Figure 4.11, we use the operations presented before in Section 4.3.1 to transform it into an optimal schedule. For any two non-overlapping flexis, we try to shrink one by $t$ and stretch the other by $t$ for the maximum possible time $t > 0$. We repeat this until there is at most one flexi left.*

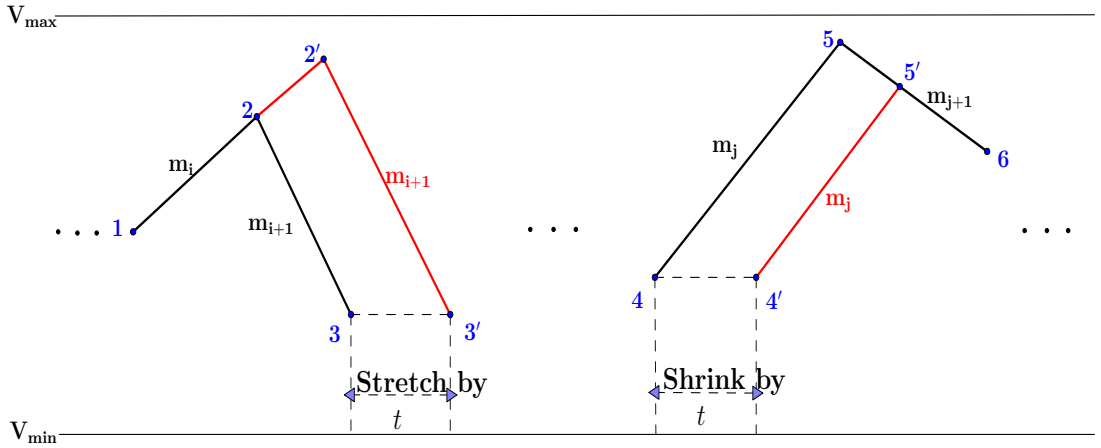



FIGURE 4.10: Shrink and stretch operations being applied to two up-down flexis.

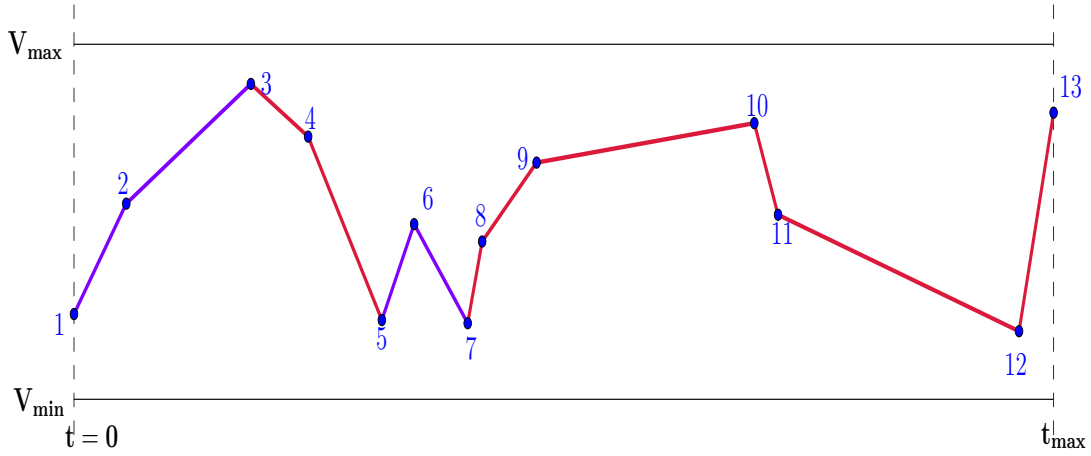



FIGURE 4.11: The original safe-schedule.

*Step 1: Here, we start off by shrinking flexi 1-2-3 (of type up-up) and stretching flexi 5-6-7 (of type up-down). This will result in straightening the 1-2-3 flexi and removal of its midpoint 2. The result is shown in Figure 4.12.*

*Step 2: Next, for the schedule shown in Figure 4.12, we will apply the procedure to flexis 2-3-4 (of type down-down) and 4-5-6 (of type up-down). This will result in straightening the 2-3-4 flexi and removal of its midpoint 3. we can see the end result in Figure 4.13.*

*Step 3: Next, in the schedule shown by Figure 4.13, we will apply the procedure to flexis 5-6-7 (of type up-up) and 8-9-10 (of type down-down). This will result in straightening of the 5-6-7 flexi and removal of its midpoint 6 (we can see the end result in the Figure 4.14).*

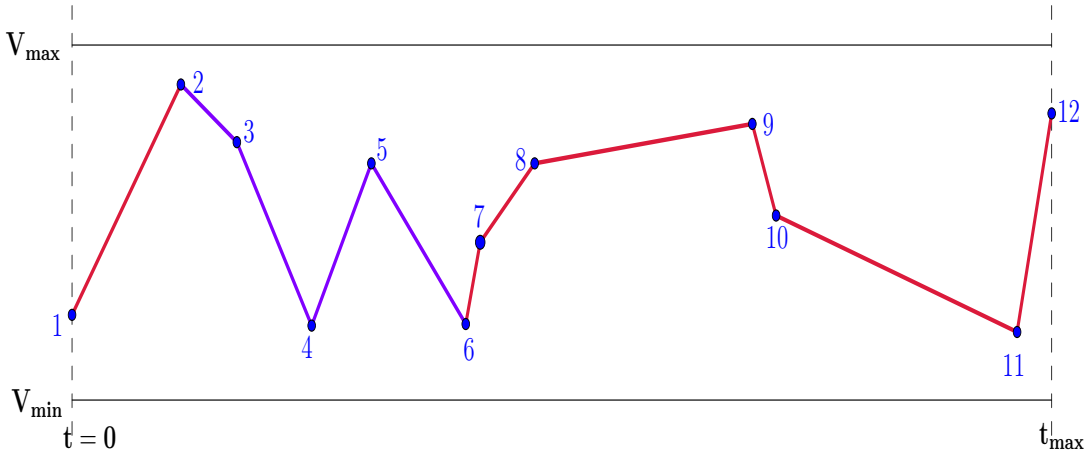



FIGURE 4.12: The resulting safe-schedule after step 1.

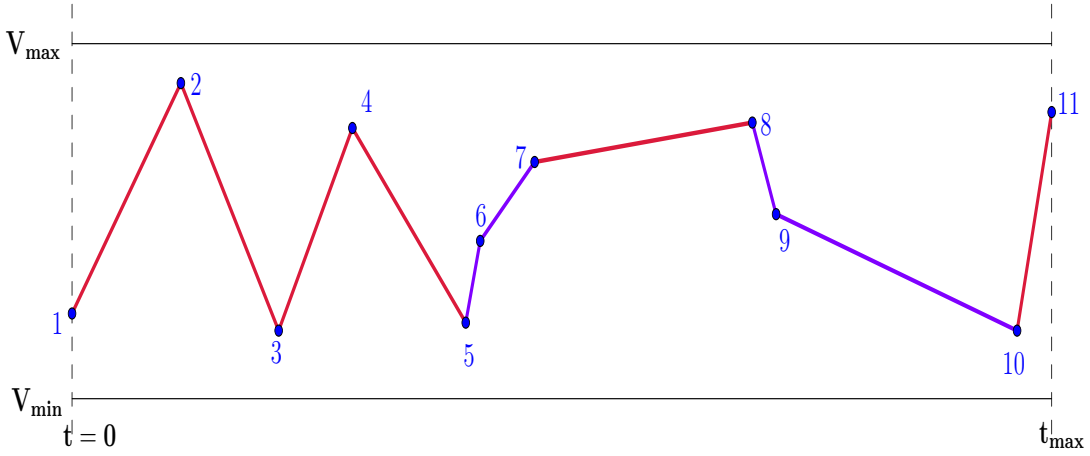



FIGURE 4.13: The resulting safe-schedule after step 2.

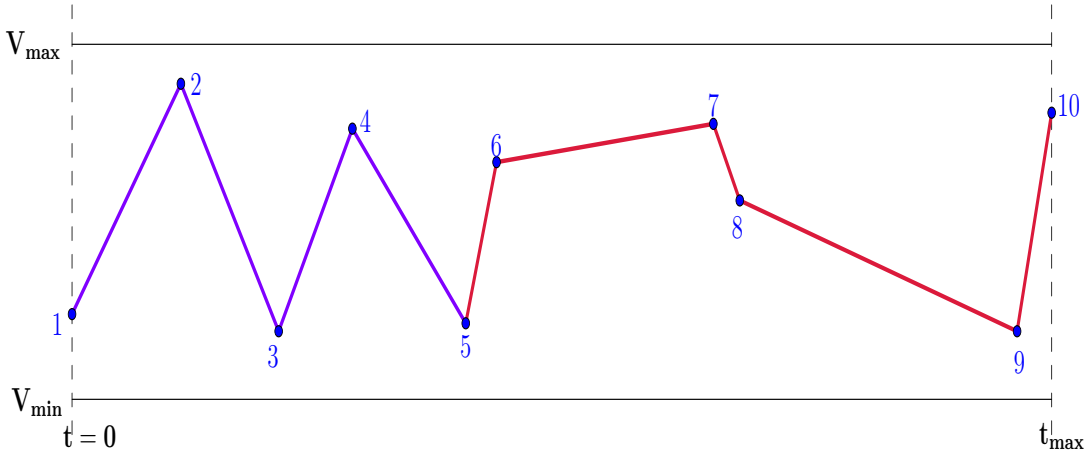



FIGURE 4.14: The resulting safe-schedule after step 3.

*Step 4: Next, we will apply the procedure to flexis 1-2-3 (of type up-down) and 3-4-5 (of type up-down) shown in Figure 4.14. This will result in moving the midpoint 2 up until it reaches $V_{max}$. The resulting safe-schedule is shown in Figure 4.15.*

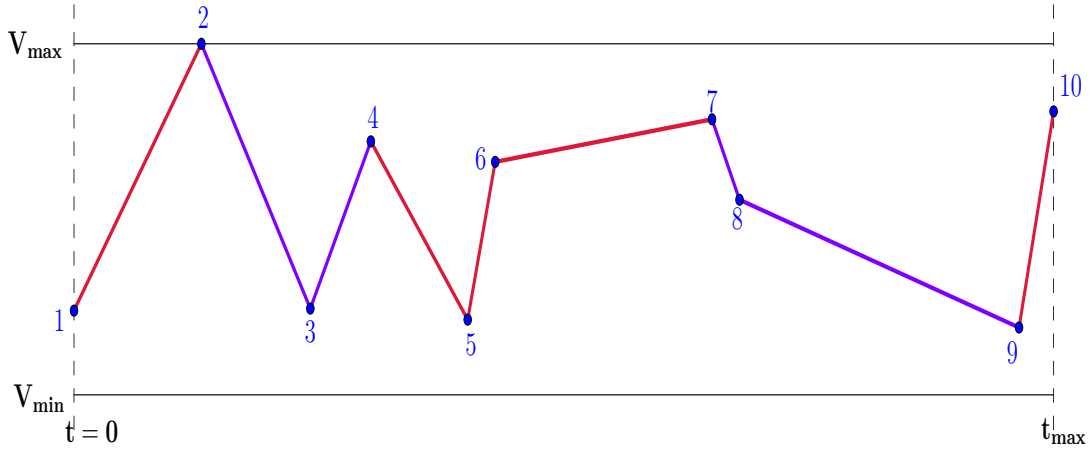



FIGURE 4.15: The resulting safe-schedule after step 4.

*Step 5: Next, in the safe schedule shown in Figure 4.15, we will apply the procedure to flexis 2-3-4 (of type down-up) and 7-8-9 (of type down-down). This will result in moving the midpoint 3 down until it reaches $V_{min}$. The resulting safe schedule is shown in Figure 4.16.*

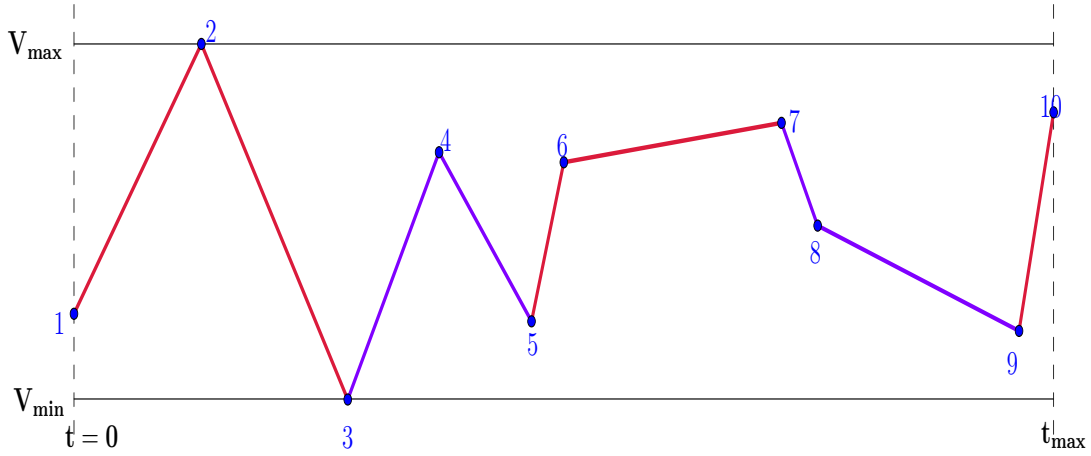



FIGURE 4.16: The resulting safe-schedule after step 5.

*Step 6: Next, in the safe schedule shown in Figure 4.16, we will apply the procedure to flexis 3-4-5 (of type up-down) and 7-8-9 (of type down-down). This will result in moving the midpoint 4 up until it reaches $V_{max}$. The output safe schedule from this step is shown in Figure 4.17.*

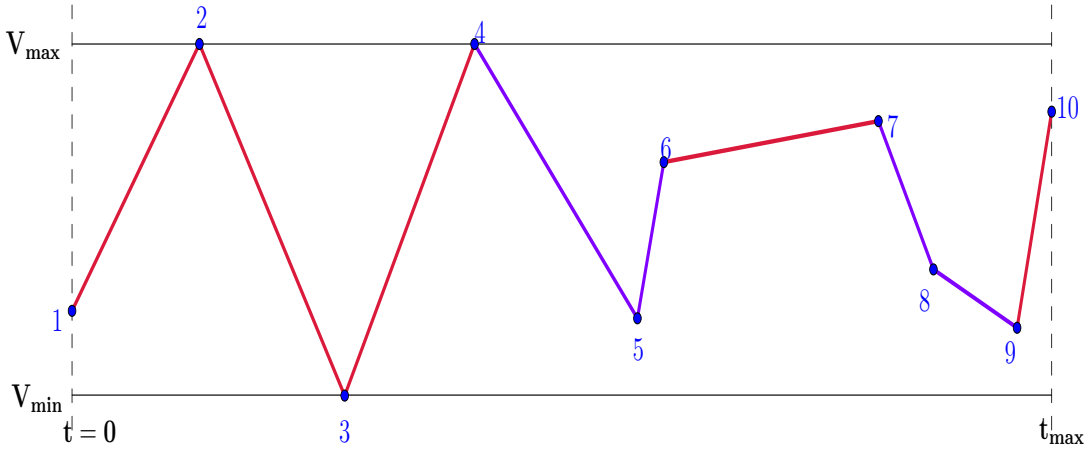



FIGURE 4.17: The resulting safe-schedule after step 6.

*Step 7: Next,in the safe schedule shown in Figure 4.17, we will apply the procedure to flexis 4-5-6 (of type down-up) and 7-8-9 (of type down-down). This will result in moving the midpoint 5 down until it reaches $V_{min}$. The resulting safe schedule is shown in Figure 4.18.*

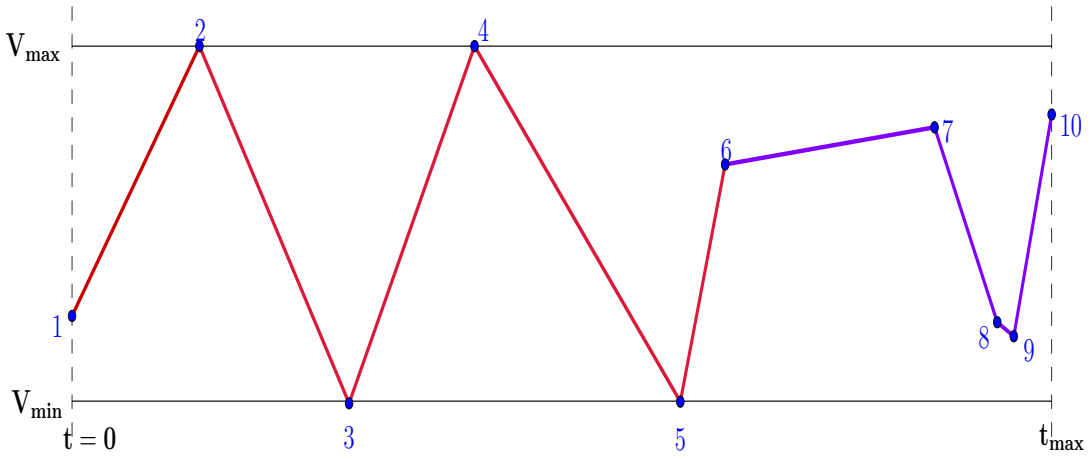



FIGURE 4.18: The resulting safe-schedule after step 7.

*Step 8: Next, in the schedule shown by Figure 4.18, we will apply the procedure to flexis 6-7-8 (of type up-down) and 8-9-10 (of type down-up). This will result in straightening of the 8-8-10 flexi and removal of the midpoint 9. The new safe-schedule is shown in Figure 4.19.*

*Step 9: Next, in the schedule shown by Figure 4.19 we will apply the procedure to flexis 5-6-7 (of type up-up) and 7-8-9 (of type down-up). This will result in moving the midpoint 8 down until it reaches $V_{min}$. The resulting safe-schedule is shown in figure 4.20.*

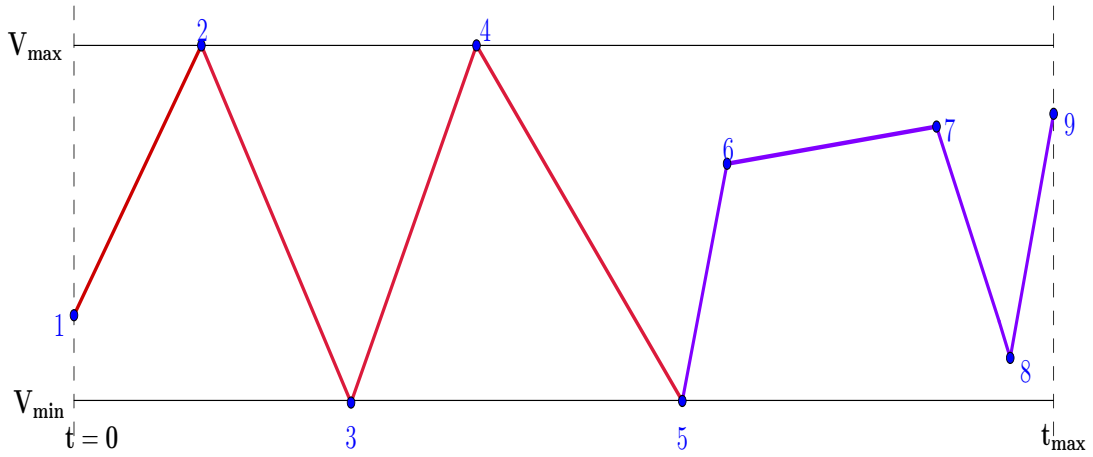



FIGURE 4.19: The resulting safe-schedule after step 8.

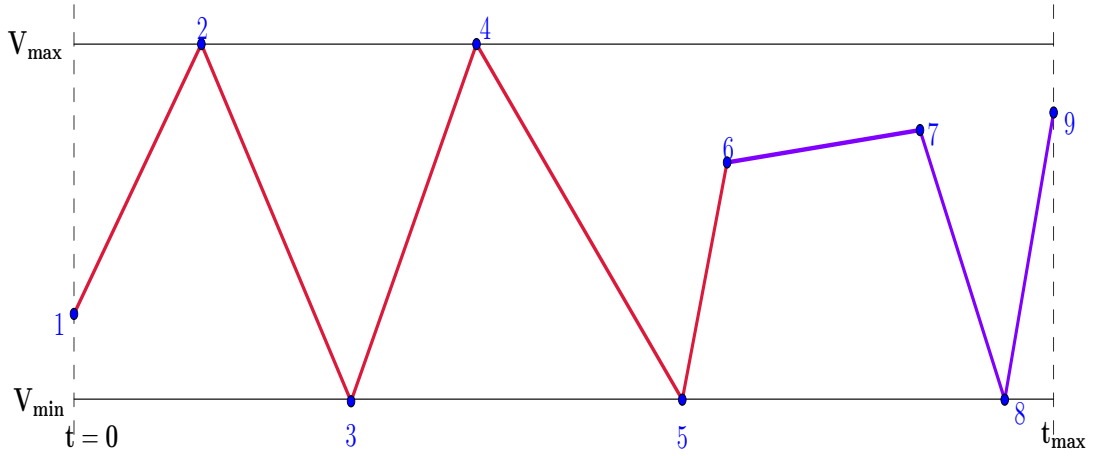



FIGURE 4.20: The resulting safe-schedule after step 9.

*Step 10:* *Since there no more non-overlapping flexis in the schedule shown in Figure 4.20, we try to move the one that remains in the leaps section to the end of the schedule. In this case, as all of them are already located after the leaps section, this step is skipped. Next, we will apply the same procedure but with the first timed action if it is a flat one or with the last timed action if it does not reach neither $V_{min}$ nor $V_{max}$ (and so shrink and stretch operations can be applied to it). In this case we apply this operation to flexi 6-7-8 (of type up-down) and the last timed action 8-9. This results in moving point 9 up until it reaches $V_{max}$. The resulting safe schedule is shown in Figure 4.21.*

*Step 11:* *Our schedule shown in Figure 4.21 is already partitioned into three distinct sections: head, leaps, and tail. However, the tail section does not follow any of the 10 patterns in Figure 4.8. We cannot apply shrink/stretch operations because the flexes 5-6-7 and 6-7-8 are overlapping. At the same time points 6 and 7 still have some flexibility in them. We apply the wedge operation to*

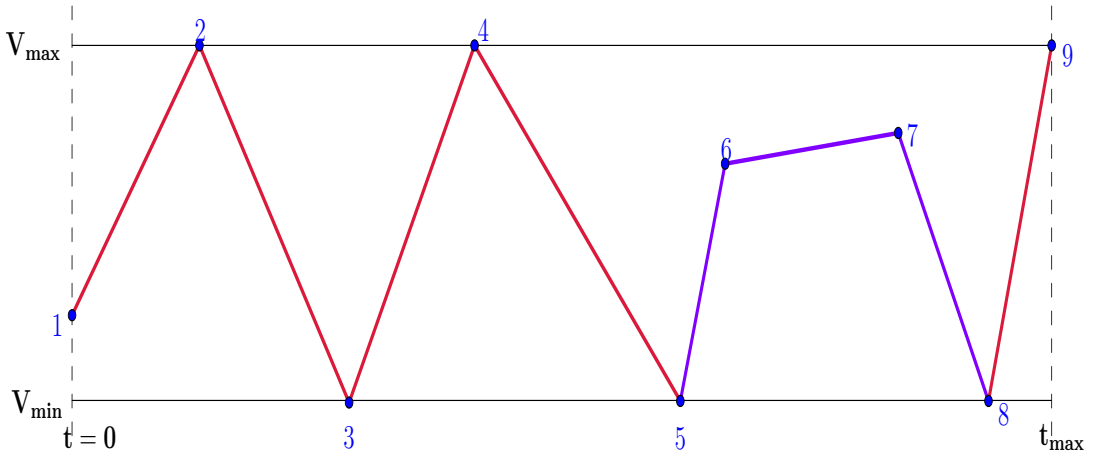



FIGURE 4.21: The resulting safe-schedule after step 10.

*the 5-6-7-8 segment to resolve this. In this case, points 6 and 7 are moved up until one of them reaches $V_{max}$ and the first one to do so is point 7. The resulting safe schedule is shown in Figure 4.22.*

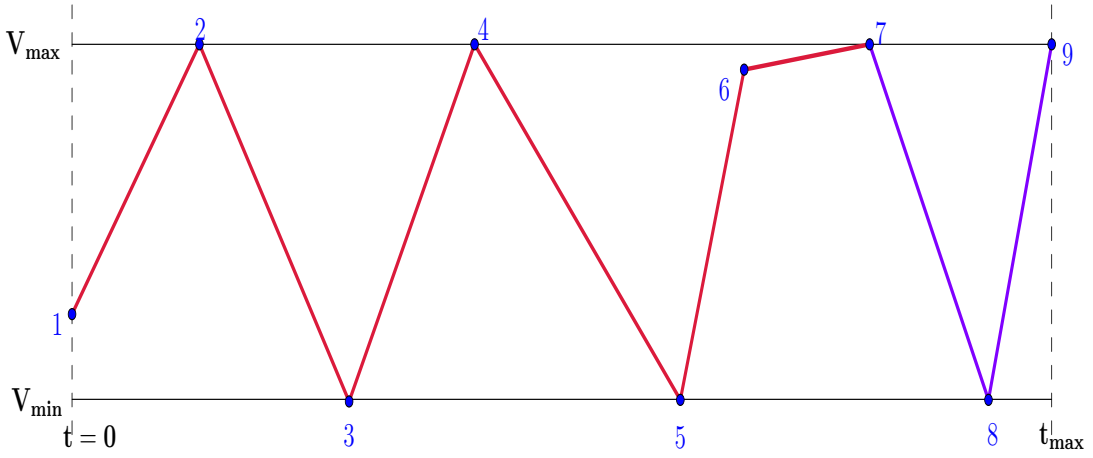



FIGURE 4.22: The resulting safe-schedule after step 11.

*Step 12: In the schedule shown by Figure 4.22, there is only one point between $V_{min}$ and $V_{max}$ left (point 6), but the tail still does not follow any of the 10 patterns. We use the shift-down operation to segment 7-8-9 and move it after 5. The resulting schedule is shown in Figure 4.23 with both the head section (1-2-3) and tail section (7-8-9) follows one of the standard patterns. The head section follows the partial-up+down pattern (Figure 4.7(e)) and the tail section follows partial-up+up pattern (Figure 4.8(b)). The leaps section (3-4-5-6-7) consists of two (complete) leaps. So, there exists an optimal schedule that has the same shape as in Figure 4.23.*

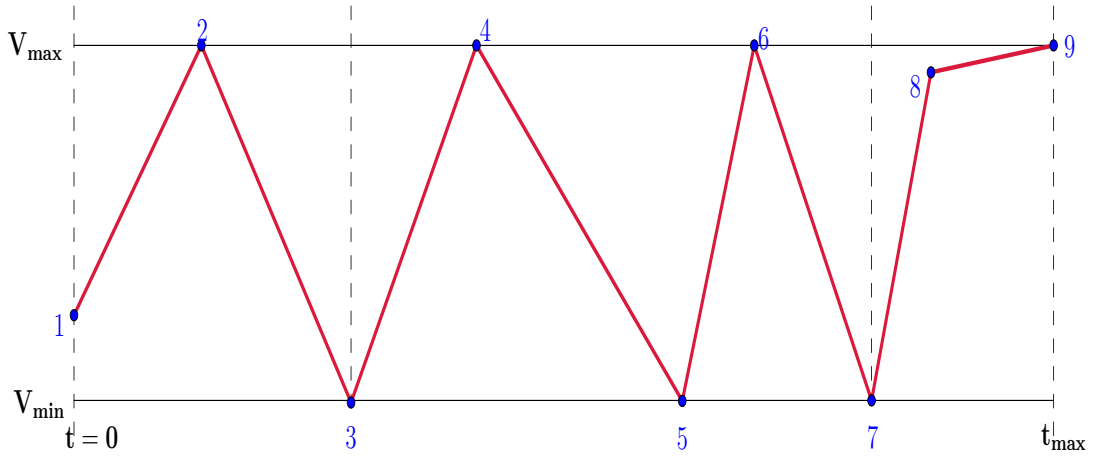



FIGURE 4.23: The resulting optimal safe-schedule after step 12.

## 4.4 Complexity of Optimal Control in One-dimension

We start with considering the easy case of infinite time horizons, before turning to the interesting case of finite time horizons.

### 4.4.1 Infinite Time Horizon

First let us consider the case $M^0 = \emptyset$. If also $M^+ \times M^- = \emptyset$ then there are no safe schedules with infinite horizon at all. Otherwise, let $(i', j') = \operatorname{argmin}_{(i,j)\in M^+\times M^-} \Delta\pi_{i,j}/\Delta t_{i,j}$. Let us pick any mode $m^- \in M^-$ and denote $t^- := (V_{\min} - V_0)/A(m^-)$. Consider the infinite schedule $\sigma$, which starts with the timed action $(m^-, t^-)$ followed by infinitely many complete leaps of type $(i', j')$. Obviously, at all times $t = t^- + k\cdot\Delta t_{i',j'}$ where $k \in \mathbb{N}$, $\sigma$ is more expensive by at most $\pi_d(m^-)+\pi_c(m^-)t^-$ from the cheapest schedule with time horizon $t$. Consequently, as $k \to \infty$, this shows that the limit superior of the average cost cannot be smaller than $\Delta\pi_{i',j'}/\Delta t_{i',j'}$. At the same time, $\sigma$ realises this long-time average.

If $M^0 \neq \emptyset$, then let $m' = \min_{m\in M^0} \pi_c(m)$ be the zero-mode with the lowest continuous cost to run. We claim that if $\pi_c(m') < \Delta\pi_{i',j'}/\Delta t_{i',j'}$ or $M^+ \times M^- = \emptyset$ then an optimal safe schedule is simply $(m', \infty)$, whose limit-average cost is $\pi_c(m')$, and otherwise $\sigma$ defined above is an optimal safe schedule. This is because, if $\pi_c(m') < \Delta\pi_{i',j'}/\Delta t_{i',j'}$, then, at any time point of $\sigma$ where a leap of some type $(i, j)$ is used, removing this leap and increasing the time $m'$ is used for by $\Delta t_{i,j}$ reduces the total cost up to this time point.

Taking into account that $\operatorname{argmin}_{(i,j)\in M^+\times M^-} \Delta\pi_{i,j}/\Delta t_{i,j}$ can be computed using logarithmic space (because multiplication, division and comparison can be [19]) we get the following theorem.

**Theorem 4.10.** *An optimal safe infinite schedule for one-dimensional multi-mode systems can be computed in deterministic* LogSpace.

### 4.4.2 Finite Time Horizon

As our one-dimensional model strictly generalises the simple linear hybrid automata considered in Chapter 3, referring to Theorem 3.7, we immediately obtain the following result.

**Theorem 4.11** (follows from Theorem 3.7)**.** *Given (one-dimensional) multi-mode system $\mathcal{A}$, constants $t_{max}$ and $C$ (both in binary), checking whether there exists a safe schedule in $\mathcal{A}$ with time horizon $t_{max}$ and total cost at most $C$ is NP-hard.*

So, we know that the decision problem for optimal schedules in one-dimensional multi-mode systems is at least NP-hard. Here, we show that the problem is NP-complete by showing that an optimal schedule exists and that each section of an optimal schedule can be guessed.

**Theorem 4.12.** *For any one-dimensional multi-mode systems $\mathcal{A}$ and $t_{max} \geq 0$, there exists an optimal schedule with time horizon $t_{max}$, and checking for the existence of an optimal schedule with cost $\leq C$ is* NP*-complete. (When $t_{max}$ and $C$ are given in binary.)*

*Proof.* First, we can simply iterate over all schedules of length one and directly calculate their costs. Next, we can iterate over pairs of modes, $m_1$ and $m_2$, and for each of them solve a linear program (LP) which will give us the cheapest schedule of length two using these two modes. This LP finds the cheapest partition of $t_{\max}$ between the two modes and has the following form:

$$\text{Minimise } \pi_c(m_1)t_1 + \pi_c(m_2)(t_{\max} - t_1) + \pi_d(m_1) + \pi_d(m_2) \text{ Subject to: } 0 \leq t_1 \leq t_{\max},$$
$$V_{\min} \leq V_0 + A(m_1)t_1 \leq V_{\max} \text{ and } V_{\min} \leq V_0 + A(m_1)t_1 + A(m_2)(t_{\max} - t_1) \leq V_{\max}.$$

This can be done in $\mathcal{O}(|\mathcal{A}|^2)$ time.

Now, for schedules of length at least three, we showed in Section 4.3 that any such a schedule can be transformed without increasing its cost into one that can be split into three sections: the head section, the leaps section, and the tail section (some of which may be empty). Due to Theorem 4.9, there are 44 combined patterns for the tail and head sections. Note that, when considering only the cost of the whole schedule, it suffices for us to know the number of leaps of each type in the leaps section and not their precise order. Notice that a schedule with time horizon $t_{\max}$ can contain at most $\lfloor t_{\max}/\Delta\pi_{i,j} \rfloor$ leaps of type $(i, j)$. The size of this

number is polynomial in the size of the input $\mathcal{A}$. There are $\mathcal{O}(|M|^2)$ types of leaps so the number of leaps of each type and the combined pattern of the schedule can be guessed non-deterministically with polynomially many bits. This guess uniquely determines the cost of the schedule. This is because, after the total time of the leaps section is deducted from $t_{\max}$, we get the exact time the head and tail section have to last for. Each combined pattern has at most one of the following: a flexi, a zero-mode, or the last state not at the border. The time remaining will determinate exactly (if at all possible) the value of this single flexible point along this schedule. Now, computing the cost of the resulting schedule and checking whether it is lower than $C$ can be done in polynomial time by guessing the modes we use and when we use them. This shows that the problem is in NP. It also shows that optimal schedules exist, because there are only finitely many options to choose from. □

## 4.5 Approximate Optimal Control in One-Dimension

### 4.5.1 Constant Factor Approximation

We show here an approximation algorithm with a 3-relative performance for the cost minimisation problem in one-dimensional multi-mode systems, which runs in $\mathcal{O}(|\mathcal{A}|^7)$ time. Our algorithm ,if the schedule is of length 2, checks all the possible pairs that cover the time $t_{max}$ and selects the cheapest. If the schedule's length is more than 2, thanks to Theorem 4.9, the schedule follows the shape of (head + leaps + tail) and the maximum length of the (head + tail) part is 5 which means that the algorithm tries all possible patterns as in the optimal schedule and for the leaps section always picks leaps of the same type. It then adds, if necessary or for cost efficiency, a partial leap to the leaps section and minimises the total cost of the just constructed schedule by optimising the time duration of this partial leap. This constant approximation algorithm is crucial for showing the existence of an FPTAS for the same problem in the next subsection.

**Theorem 4.13.** *Computing a safe schedule with total cost at most three times larger than the optimal one for one-dimensional multi-mode system $\mathcal{A}$ can be done in $\mathcal{O}(|\mathcal{A}|^7)$ time.*

*Proof.* First, we iterate over all possible schedules of length at most two and find the cheapest one. Next, thanks to Theorem 4.9, all optimal schedules longer than two can be transformed into one of 44 different allowable patterns. Our algorithm will simply iterate over all 44 possible combined patterns for the head and tail

section and within each pattern over all allowed combinations of modes that make up the head and tail sections. (Due to Theorem 4.9 we know there are at most five actions in each pattern.) As for the leaps section, the algorithm iterates over all possible leap types. The algorithm then finds the cheapest schedule that uses only leaps of the selected type (and potentially a partial leap if necessary). This can be done in a constant number of arithmetic operations. For instance, for the head pattern up+down (e), with $m_1, m_2$ as modes, and tail pattern partial-up+down+up (h), with $m_3, m_4, m_5$ as modes, and all leaps of type $(i, j)$, the algorithm does the following. Let $t^* = t_{\max} - (V_{\max} - V_0)/A(m_1) - \Delta t_{m_2} - \Delta t_{m_5}$. Intuitively, $t^*$ is the total time left for all the leaps and the remaining partial-up+down part whose exact timing is flexible. Note that the partial-down+up part is a partial leap of type $(m_3, m_4)$ and its cost can be expressed as $\pi_d(m_3) + \pi_d(m_4) + \alpha t$, where $\alpha = (\Delta\pi_{m_3,m_4} - \pi_d(m_3) - \pi_d(m_4))/\Delta t_{m_3,m_4}$ and $t \leq \Delta t_{m_3,m_4}$ is the total amount of time the partial-down+up part takes. Now if $\alpha < \Delta\pi_{i,j}/\Delta t_{i,j}$ then the number of leaps should be as small as possible, i.e. $\lfloor (t^* - \Delta t_{m_3,m_4})/\Delta t_{i,j} \rfloor$, to minimise the total cost. Otherwise, the number of leaps has to be the largest possible, i.e. $\lfloor t^*/\Delta t_{i,j} \rfloor$.

If the addition of a partial leap of type $(i, j)$ is necessary, then we need to compare $\alpha$ with $\beta = (\Delta\pi_{i,j} - \pi_d(i) - \pi_d(j))/\Delta t_{i,j}$, which is the cost per time unit of using a partial leap of type $(i, j)$ disregarding its discrete cost (as it is already paid for anyway). If $\alpha < \beta$ then the partial leap of type $(i, j)$ has to be the biggest possible to minimise the cost, which results in either this partial leap becoming complete (i.e. a leap) or the partial-up+down part disappearing, so changing the tail to the up (e) pattern. Otherwise, if $\alpha \geq \beta$, the partial leap of type $(i, j)$ has to be the smallest possible to minimise the cost, which results in either this partial leap disappearing or the partial-up+down part turning into a (complete) leap of type $(m_3, m_4)$. Essentially in the same way we can deal with the remaining combined patter. In the end, we compare the total costs of all the constructed schedules and return the cheapest one. It is easy to see now that this iterative algorithm runs in time $\mathcal{O}(|\mathcal{A}|^7)$. If the schedule is of length 2, we check all the possible pairs in time $\mathcal{O}(|\mathcal{A}|^2)$ and select the cheapest. If the schedule's length is more than 2, the schedule follows the shape of (head + leaps + tail). The maximum length of the (head + tail) part is 5 which means that we go throw all the possible combinations in time $\mathcal{O}(|\mathcal{A}|^5)$. For the leap part, we always use leaps of the same time which means that we go throw all the possible leaps in time $\mathcal{O}(|\mathcal{A}|^2)$. Now, the only thing left to do is to show that its performance ratio is 3.

Let $\sigma^*$ be an optimal schedule with time horizon $t_{\max}$. Let us focus again on the up+down pattern with modes $m_1, m_2$ and the head partial-up+down+up pattern

with modes $m_3, m_4, m_5$ for the tail; the reasoning is almost the same in all the other cases. If $\sigma^*$ has no leaps at all then due to Theorem 4.9 it can have at most five timed actions and our algorithm will return the optimal solution, because it tries all such possibilities. So now assume that the cheapest leap (per time unit as measured by $\Delta\pi_{i,j}/\Delta t_{i,j}$) in $\sigma^*$ has type $(i,j)$. Let us consider the iteration of our approximation algorithm when the pattern and modes mentioned above, and type $(i,j)$ for the leaps section are considered. Let $\sigma$ be the schedule constructed by our algorithm in this iteration. We will show that $\pi(\sigma) - \pi(\sigma^*) \leq 2 \cdot \Delta\pi_{i,j}$. and $\sigma$. Notice that both $\sigma$ and $\sigma^*$ use modes $m_1, m_2$ and $m_3$ for exactly the same time in their head and tail sections, respectively, so we do not need to consider these. Hence, the difference between them can only be in the amount of time these schedules dedicate to the partial-up+down part (partial leap) that uses modes $m_4$ and $m_5$ and to the leaps section. Let us denote by $t_l^*$, $t_e^*$ the time duration of the leaps section and the partial-up+down part, respectively, in $\sigma^*$. Similarly, let $t_l$ and $t_e$ be these time durations in $\sigma$. Note that $t_l + t_e = t_l^* + t_e^*$. We claim that $|t_e^* - t_e| < \Delta t_{i,j}$. If it was the case that $t_e^* - t_e \geq \Delta t_{i,j}$ then we could have reduced the total cost of $\sigma^*$ by shrinking the partial-up+down part by $\Delta t_{i,j}$ and adding another type $(i,j)$-leap to $\sigma^*$. This is because, the shrinking operation is safe for any value from some closed interval (see Section 4.3.1), and we already know that shrinking can be safely done for $t_e^* - t_e$ which is $\geq \Delta t_{i,j}$. At the same time, the cost of $\sigma^*$ would become lower, because otherwise the way $\sigma$ was chosen would imply that one less leap of type $(i,j)$ should be in $\sigma$. Similarly we can reason that if it was the case that $t_e - t_e^* \geq \Delta t_{i,j}$ then we could have reduced the total cost of $\sigma^*$ by stretching the partial-up+down part by $\Delta t_{i,j}$ and removing a leap of type $(i,j)$ from $\sigma^*$ (we know at least one exists in $\sigma^*$).

Note that the cost of the leap section in $\sigma$ is the same or lower than in $\sigma^*$ up to time point $t_\$ = \lfloor \min\{t_l, t_l^*\}/\Delta t_{i,j} \rfloor \cdot \Delta t_{i,j}$, because the cheapest leap in $\sigma^*$ is of type $(i,j)$.

If $t_l \geq t_l^*$ then we know that $t_l - t_\$ \leq 2 \cdot \Delta t_{i,j}$, so the cost of the leap section is at most $2 \cdot \Delta\pi_{i,j}$ more expensive in $\sigma$. At the same time, $t_e \leq t_e^*$ holds. The partial-up+down part therefore costs at least as much in $\sigma^*$ as it does in $\sigma$. On the other hand, if $t_l < t_l^*$, then we know that the cost of the leaps section in $\sigma^*$ has to be at least $\lfloor t_l^*/\Delta t_{i,j} \rfloor \cdot \Delta\pi_{i,j}$. At the same time, the total cost of $\sigma$ is at most $\lfloor t_l/\Delta t_{i,j} \rfloor \cdot \Delta\pi_{i,j} + \Delta\pi_{i,j}$, where the last compound is the maximum cost of a partial leap. Clearly the difference between these two costs is at most $\Delta\pi_{i,j}$. As for the partial-up+down part in $\sigma$ we claim that that $c_e^* + \Delta\pi_{i,j} \leq c_e$. Recall that the length of $t_e$ was picked in a way to minimise the cost. Therefore, if $c_e^* + \Delta\pi_{i,j} > c_e$ then picking $t_e^*$ for the time duration of the partial-up+down part in $\sigma$ would have

lowered the total cost. This time duration is achievable, because shrinking this part by $t_e - t_e^* \leq \Delta t_{i,j}$ while extending or introducing a (partial) leap of type $(i, j)$ is possible.

The two estimates, for the leaps section and partial-up+down part, give us that $\pi(\sigma^*) - \pi(\sigma) \leq 2 \cdot \Delta\pi_{i,j}$. At the same time, $\pi(\sigma^*) \geq \Delta\pi_{i,j}$ holds, which shows that $\sigma$ is at most three times more expensive that $\sigma^*$. Note that the longer the leaps section is in the optimal schedule, the better the performance ratio, e.g. if $\lfloor t_l^*/\Delta t_{i,j} \rfloor = k$ then the performance ratio is $1 + 2/k$. □

### 4.5.2 FPTAS Algorithm

We show here that the cost minimisation problem for the general case one dimensional multi-mode systems is in FPTAS by a polynomial time reduction to the 0-1 Knapsack problem, for which many FPTAS algorithms are available (see e.g. [32]). This is similar to the FPTAS construction in Section 3.6.2, but differs in how the modes with fractional duration are handled. First we iterate over all possible schedules of length at most two and find the cheapest one in polynomial time. Next, thanks to Theorem 4.9, all optimal schedules longer than two can be transformed into one of 44 different patterns. Each of these patterns results in a slightly different FPTAS formulation. An FPTAS for the general model consists of all of these individual FPTASes executed one after another.

**Theorem 4.14.** *Solving the optimal control problem for multi-mode systems with relative performance $\rho$ takes $\mathcal{O}(\mathsf{poly}(1/\rho)\mathsf{poly}$(size of the instance)) time and is therefore in FPTAS.*

*Proof.* We consider here only one of the 44 possible pattern cases, because all these FPTAS algorithms will look essentially the same with a little bit difference which will be explained in Section 4.5.2. We combine all these FPTASes into a single FPTAS for the general model by running them one by one. The case we will look at is up+down pattern, with modes $m_1, m_2$, for the head section and partial-up+up+down, with modes $m_3, m_4, m_5$, for the tail section. The schedule is shown by Figure 4.24. We consider all combinations of these five modes $m_i$ individually and select the cheapest one, and therefore consider them given. (Note that there are only quintically many such combinations.) W.l.o.g. we assume that $\Delta t_{m_3} > \Delta t_{m_4}$, because otherwise we could swap the role of $m_3$ with $m_4$ in our algorithm below. Note that any schedule with this pattern which picks $m_3$ in the tail for $\alpha\Delta t_{m_3}$ amount of time, uses $m_4$ for $(1-\alpha)\Delta t_{m_4}$ amount of time in the tail section.

Let $c^*$ be the 3-approximation, which can be computed using the procedure from Theorem 4.13, of the optimal cost $o^*$. To get an approximation to our optimal cost problem with a relative performance $\rho$, it suffices to find a solution with $c^*\rho/3$ absolute performance. We split this into two equal parts of $\epsilon = c^*\rho/6$. An optimal solution to the knapsack instance that we produce will provide us with a schedule with cost no greater than $\epsilon$ over the optimal one. Moreover, a solution to the knapsack instance with $\delta$ absolute error will provide a schedule with an $\epsilon + \delta$ absolute error. Therefore, it suffices to set $\delta = \epsilon$ to find a schedule with $\rho$ relative performance. In our reduction, the total value of all the items in the resulting knapsack problem is at most $4|M|^2$ times the optimal cost for safe schedules, so by using $\rho' = \rho/(12|M|^2)$ for the resulting knapsack problem we will find a near optimal solution with a relative performance $\rho$ for multi-mode systems. The running time of this procedures is $\mathcal{O}(\mathsf{poly}(1/\rho)\mathsf{poly}(|M|)\mathsf{poly}(\text{size of the knapsack instance}))$. This suffices to establish the inclusion of the cost minimisation problem for multi-mode systems in FPTAS.

For each type of leaps, $(m, m') \in M^+ \times M^-$, we build the following items for this knapsack problem instance: $\{(2^i \cdot \Delta t_{m,m'}, 2^i \cdot \Delta\pi_{m,m'}) \mid i \in \mathbb{N} \wedge 2^i \cdot \Delta\pi_{m,m'} \leq c^* \wedge 2^i \cdot \Delta t_{m,m'} \leq t_{\max}\}$. Let $i^* \in \mathbb{N}$ be smallest such that $0 \leq 2^{-i^*} \cdot (\Delta\pi_{m_3} - \pi_d(m_3) - \Delta\pi_{m_4} + \pi_d(m_4)) \leq \epsilon$. For both $m_3$ and $m_4$ we add the following extra multiset of items: $\{(2^{-i} \cdot (\Delta t_{m_3} - \Delta t_{m_4}), 2^{-i} \cdot (\Delta\pi_{m_3} - \pi_d(m_3) - \Delta\pi_{m_4} + \pi_d(m_4))) \mid i \in \mathbb{Z}_+ \wedge i \leq i^* \wedge 2^{-i} \cdot (\Delta\pi_{m_3} - \pi_d(m_3) - \Delta\pi_{m_4} + \pi_d(m_4)) \leq c^*\}$ and additionally $(2^{-i^*} \cdot (\Delta t_{m_3} - \Delta t_{m_4}), 2^{-i^*} \cdot (\Delta\pi_{m_3} - \pi_d(m_3) - \Delta\pi_{m_4} + \pi_d(m_4))$, which is a copy of an element already in the multiset. The reason behind using this constraint $2^{-i} \cdot (\Delta\pi_{m_3} - \pi_d(m_3) - \Delta\pi_{m_4} + \pi_d(m_4)) \leq c^*$ is that we do not want to add fractions with the modes that are very expensive to run which exceeds the cost boundary $c^*$. The removing of these unnecessary items increases the algorithm performance as we consider solving the knapsack problem over a smaller time horizon which is calculated using equation 4.4. Note that this models the fact that the more $m_3$ is used in the tail section the less mode $m_4$ is used in tail section and with the same proportion. Note that because of the assumption that $\Delta t_{m_3} \geq \Delta t_{m_4}$, we are sure that all the items produced have a positive time duration. Let $t_\Sigma$ be the time span of all items in this knapsack instance. We set the volume of this 0-1 knapsack instance to be

$$t_\Sigma - t_{\max} + (V_{\max} - V_0)/A(m_1) + \Delta t_{m_2} + \Delta t_{m_4} + \Delta t_{m_5} \tag{4.1}$$

The just produced knapsack problem has the following properties:

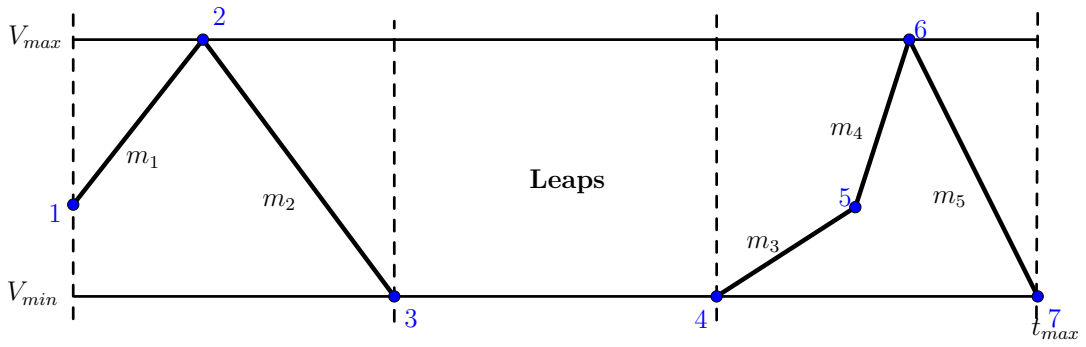



FIGURE 4.24: FPTAS case with up+down pattern for the head section and partial-up+up+down for the tail section.

- the size of its description is polynomial in the size of the original problem including the relative performance;
- fractional time duration of $m_3$ in the tail section can be overestimated by joining together the fractional items for both $m_3$ and $m_4$ (which do not include discrete costs), so that we do not exceed the volume by $2^{-i^*} \cdot (\Delta t_{m_3} - \Delta t_{m_4})$ or more;
- $n$ leaps of of type $(m, m')$ in $\sigma$ can be achieved by picking the items for this type and corresponding to the binary representation of $n$; and
- The volume of these items is $\geq t_{\max} - (V_{\max} - V_0)/A(m_1) - \Delta t_{m_2} - \Delta t_{m_4} - \Delta t_{m_5}$, which leaves enough space for modes $m_1$ and $m_2$ in the head section, and mode $m_5$ and the minimum amount of time for the $m_3 + m_4$ part (when $\alpha = 0$) in the tail section. Let $v^*$ be the value of the items in this knapsack and $o^*$ denotes the optimal cost. Then

  $$0 \leq v^* + \pi_d(m_1) + \pi_c(m_1)(V_{\max} - V_0)/A(m_1) + \Delta\pi_{m_2} + \pi_d(m_3) + \Delta\pi_{m_4} + \Delta\pi_{m_5} - o^* \leq \epsilon$$

  The value $v^*$ contains the leaps part as well as the fractions of the mode $m_3$. The cost of the fraction is the difference between the running cost of $m_3$ and $m_4$ (without the discrete costs) which is always positive because of the assumption we stated before. So, the discrete cost of $m_3$ needs to be added as well as the total cost (discrete and continuous) of $m_4$ which is denoted by $\Delta\pi_{m_4}$. Note that, we always use modes $m_1$ and $m_2$ for the head section so, the cost of using them (discrete and continuous) should be added and the same for the mode $m_5$ in the tail section.

- Let $V_\Sigma$ be the value of all items in the multiset. For any solution to the knapsack problem with value $V$ we get a schedule $\sigma'$ with cost $\leq V_\Sigma - V + \pi_d(m_1) + \pi_c(m_1)(V_{\max} - V_0)/A(m_1) + \Delta\pi_{m_2} + \pi_d(m_3) + \Delta\pi_{m_4} + \Delta\pi_{m5}$.

All of this shows that solving this knapsack instance with a relative performance of $\rho/(12|M|^2)$ gives us a safe schedule with relative performance of $\rho$. □

**FPTAS Other Cases**

In the previous section, we introduced a formal proof of the FPTAS algorithm based on a reduction into the 0-1 knapsack problem. We introduced only one case out of 44 different cases. All the cases are the same except for the part when we generate the fractions. For the case introduced before, we generated fractions of an up-up segment which means that if we use the first mode for time $\alpha\Delta t_{m_3}$ we use the second mode for time $(1-\alpha)\Delta t_{m_4}$. We did not explained how we generate fractions of a flat segment, an up-up segment or down-down segment. So, in this sub-section, we discuss further 4 cases from the 44 possible cases.

- The case we will look at is flat+up+down pattern, with modes $m_1, m_2, m_3$, for the head section and up, with mode $m_4$, for the tail section. The case is shown in Figure 4.25. We consider all combinations of these four modes $m_i$ individually, and therefore consider them given. (Note that there are only quartically many such combinations.)

  For each type of leaps, $(m, m') \in M^+ \times M^-$, we build the following items for this knapsack problem instance: $\{(2^i \cdot \Delta t_{m,m'}, 2^i \cdot \Delta\pi_{m,m'}) \mid i \in \mathbb{N} \wedge 2^i \cdot \Delta\pi_{m,m'} \leq c^* \wedge 2^i \cdot \Delta t_{m,m'} \leq t_{\max}\}$. The flexible point that can be moved in this schedule is point 2, because there exist a *flat* mode before it. We consider the possibilities of extending (until $t_{max}$) or shrinking (until 0) the timed action with a zero mode. The flat mode can be used until the end of the time horizon $t_{max}$ without affecting the schedule safety. So, let $\Delta\pi_{m_1} = \min(t_{max} \cdot \pi_c(m_1) + \pi_d(m_1), c^*)$ and $\Delta t_{m_1} = (\Delta\pi_{m_1} - \pi_d(m_1))/\pi_c(m_1)$. Let $i^* \in \mathbb{N}$ be smallest such that $2^{-i^*} \cdot (\Delta\pi_{m_1} - \pi_d(m_1)) \leq \epsilon$. For the mode $m_1$ we add the following extra multiset of items: $\{(2^{-i} \cdot \Delta t_{m_1}, 2^{-i} \cdot (\Delta\pi_{m_1} - \pi_d(m_1))) \mid i \in \mathbb{Z}_+ \wedge i \leq i^*\}$ and additionally $(2^{-i^*} \cdot \Delta t_{m_1}, 2^{-i^*} \cdot (\Delta\pi_{m_1} - \pi_d(m_1)))$, which is a copy of an element already in the multiset.

  Let $t_\Sigma$ be the time span of all items in this knapsack instance. We set the volume of this 0-1 knapsack instance to be

$$t_\Sigma - t_{\max} + (V_{\max} - V_0)/A(m_2) + \Delta t_{m_3} + \Delta t_{m_4} \tag{4.2}$$

The just produced knapsack problem has the following properties:

- The volume of the leaps part is $\geq t_{\max} - (V_{\max} - V_0)/A(m_2) - \Delta t_{m_3} - \Delta t_{m_4}$, which leaves enough space for modes $m_2$ and $m_3$ in the head section, and mode $m_4$ in the tail section. Let $v^*$ be the value of the items in this knapsack and $o^*$ denotes the optimal cost. Then

  $$0 \leq v^* + \pi_d(m_1) + \pi_d(m_2) + \pi_c(m_2)(V_{\max} - V_0)/A(m_2) + \Delta\pi_{m_3} + \Delta\pi_{m_4} - o^* \leq \epsilon$$

  The value $v^*$ contains the leaps part as well as the fractions of the mode $m_1$ (continuous cost only).
- Let $V_\Sigma$ be the value of all items in the multiset. For any solution to the knapsack problem with value $V$ we get a schedule $\sigma'$ with cost $\leq V_\Sigma - V + \pi_d(m_1) + \pi_d(m_2) + \pi_c(m_2)(V_{\max} - V_0)/A(m_2) + \Delta\pi_{m_3} + \Delta\pi_{m_4}$.

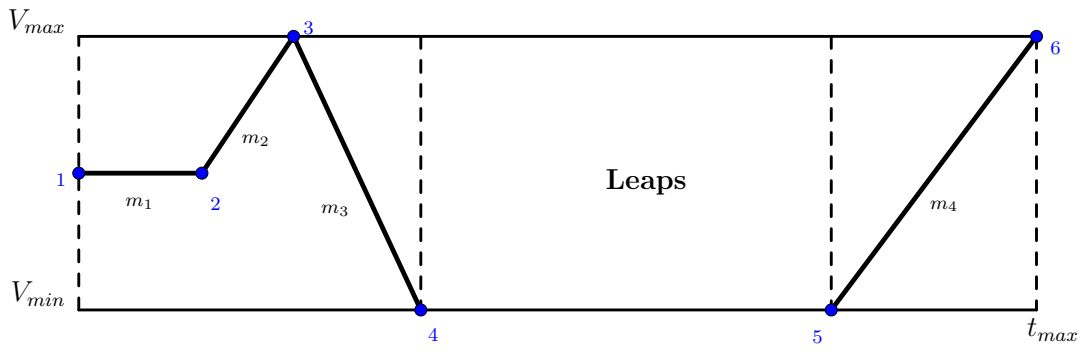



FIGURE 4.25: FPTAS case with flat+up+down pattern for the head section and up for the tail section.

- In the next case, we will look at is the up+partial-down+down pattern, with modes $m_1, m_2, m_3$, for the head section and the up pattern, with mode $m_4$, for the tail section. The case is shown in Figure 4.26.

  We consider all combinations of these four modes $m_i$ individually, and therefore consider them given. (Note that there are only quartically many such combinations.)

  For each type of leaps, $(m, m') \in M^+ \times M^-$, we build the following items for this knapsack problem instance: $\{(2^i \cdot \Delta t_{m,m'}, 2^i \cdot \Delta\pi_{m,m'}) \mid i \in \mathbb{N} \wedge 2^i \cdot \Delta\pi_{m,m'} \leq c^* \wedge 2^i \cdot \Delta t_{m,m'} \leq t_{\max}\}$.

  We assume that $\Delta t_{m_2} \geq \Delta t_{m_3}$, because otherwise we could swap the role of $m_2$ with $m_3$ in our algorithm below. Note that any schedule with this pattern which picks $m_2$ in the head for $\alpha\Delta t_{m_2}$ amount of time, uses $m_3$ for $(1-\alpha)\Delta t_{m_3}$

amount of time in the head section. Let $i^* \in \mathbb{N}$ be smallest such that $2^{-i^*} \cdot (\Delta\pi_{m_2} - \pi_d(m_2) - \Delta\pi_{m_3} + \pi_d(m_3)) \leq \epsilon$. For both $m_2$ and $m_3$ we add the following extra multiset of items: $\{(2^{-i} \cdot (\Delta t_{m_2} - \Delta t_{m_3}), 2^{-i} \cdot (\Delta\pi_{m_2} - \pi_d(m_2) - \Delta\pi_{m_3} + \pi_d(m_3))) \mid i \in \mathbb{Z}_+ \wedge i \leq i^* \wedge 2^{-i} \cdot (\Delta\pi_{m_2} - \pi_d(m_2) - \Delta\pi_{m_3} + \pi_d(m_3)) \leq c^*\}$ and additionally $(2^{-i^*} \cdot (\Delta t_{m_2} - \Delta t_{m_3}), 2^{-i^*} \cdot (\Delta\pi_{m_2} - \pi_d(m_2) - \Delta\pi_{m_3} + \pi_d(m_3)))$, which is a copy of an element already in the multiset. Note that this models the fact that the more $m_2$ is used in the head section the less mode $m_3$ is used in tail section and with the same proportion. Note that because of the assumption that $\Delta t_{m_2} \geq \Delta t_{m_3}$, we are sure that all the items produced have positive time durations. Let $t_\Sigma$ be the time span of all items in this knapsack instance. We set the volume of this 0-1 knapsack instance to be

$$t_\Sigma - t_{\max} + (V_{\max} - V_0)/A(m_1) + \Delta t_{m_3} + \Delta t_{m_4} \tag{4.3}$$

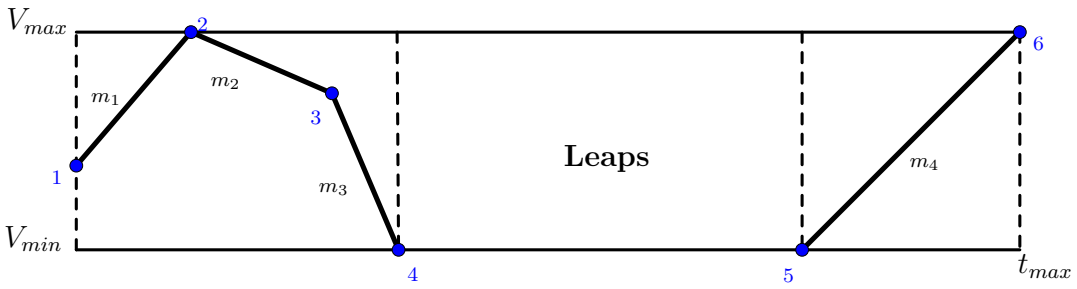



FIGURE 4.26: FPTAS case with up+partial-down+down pattern for the head section and up for the tail section.

The produced 0-1 knapsack problem has the following properties:

- fractional time duration of $m_2$ in the head section can be overestimated by joining together the fractional items for both $m_2$ and $m_3$ (which do not include discrete costs), so that we do not exceed the volume by $2^{-i^*} \cdot (\Delta t_{m_2} - \Delta t_{m_3})$ or more;
- $n$ leaps of of type $(m, m')$ in $\sigma$ can be achieved by picking the items for this type and corresponding to the binary representation of $n$; and
- The volume of these items is $\geq t_{\max} - (V_{\max} - V_0)/A(m_1) - \Delta t_{m_3} - \Delta t_{m_4}$, which leaves enough space for modes $m_1$ and the minimum amount of time for the $m_2 + m_3$ part (when $\alpha = 0$) in the head section, and mode $m_4$ in the tail section. Let $v^*$ be the value of the items in this knapsack and $o^*$ denotes the optimal cost. Then

$$0 \leq v^* + \pi_d(m_1) + \pi_c(m_1)(V_{\max} - V_0)/A(m_1) + \pi_d(m_2) + \Delta\pi_{m_3} + \Delta\pi_{m_4} - o^* \leq \epsilon$$

The value $v^*$ contains the leaps part as well as the fractions of the mode $m_2$ (with continuous cost only). .

– Let $V_\Sigma$ be the value of all items in the multiset. For any solution to the knapsack problem with value $V$ we get a schedule $\sigma'$ with cost $\leq V_\Sigma - V + \pi_d(m_1) + \pi_c(m_1)(V_{\max} - V_0)/A(m_1) + \pi_d(m_2) + \Delta\pi_{m_3} + \Delta\pi_{m4}$.

- The next case we consider is when we chose the empty pattern for the head section and partial-up+down+up pattern, with modes $m_1, m_2, m_3$, for the tail section. The case is shown by Figure 4.27. We consider all combinations of these four modes $m_i$ individually, and therefore consider them given. (Note that there are many such combinations of the third order only.)

  For each type of leaps, $(m, m') \in M^+ \times M^-$, we build the following items for this knapsack problem instance: $\{(2^i \cdot \Delta t_{m,m'}, 2^i \cdot \Delta\pi_{m,m'}) \mid i \in \mathbb{N} \wedge 2^i \cdot \Delta\pi_{m,m'} \leq c^* \wedge 2^i \cdot \Delta t_{m,m'} \leq t_{\max}\}$.

  Note that any schedule with this pattern which picks $m_1$ in the tail for $\alpha\Delta t_{m_1}$ amount of time, uses $m_2$ for $\alpha\Delta t_{m_2}$ amount of time in the tail section. Let $i^* \in \mathbb{N}$ be smallest such that $2^{-i^*} \cdot (\Delta\pi_{m_1} - \pi_d(m_1) + \Delta\pi_{m_2} - \pi_d(m_2)) \leq \epsilon$. For both $m_1$ and $m_2$ we add the following extra multiset of items: $\{(2^{-i} \cdot (\Delta t_{m_1} + \Delta t_{m_2}), 2^{-i} \cdot (\Delta\pi_{m_1} - \pi_d(m_1) + \Delta\pi_{m_2} - \pi_d(m_2))) \mid i \in \mathbb{Z}_+ \wedge i \leq i^* \wedge 2^{-i} \cdot (\Delta\pi_{m_1} - \pi_d(m_1) + \Delta\pi_{m_2} - \pi_d(m_2)) \leq c^*\}$ and additionally $(2^{-i^*} \cdot (\Delta t_{m_1} + \Delta t_{m_2}), 2^{-i^*} \cdot (\Delta\pi_{m_1} - \pi_d(m_1) + \Delta\pi_{m_2} - \pi_d(m_2))$, which is a copy of an element already in the multiset. Note that this models the fact that the less $m_1$ is used in the tail section the less mode $m_2$ is used in tail section and with the same proportion. Let $t_\Sigma$ be the time span of all items in this knapsack instance. We set the volume of this 0-1 knapsack instance to be

$$t_\Sigma - t_{\max} + \Delta t_{m_3} \tag{4.4}$$

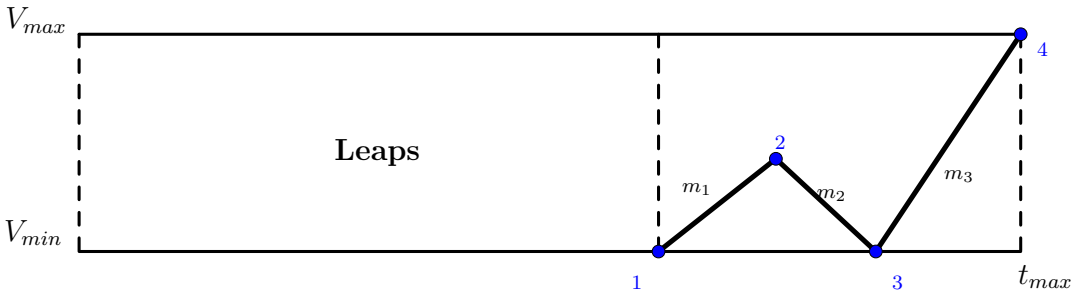



FIGURE 4.27: FPTAS case with empty pattern for the head section and partial-up+down+up for the tail section.

The produced 0-1 knapsack problem has the following properties:

  - $n$ leaps of of type $(m, m')$ in $\sigma$ can be achieved by picking the items for this type and corresponding to the binary representation of $n$; and
  - The volume of these items is $\geq t_{\max} - \Delta t_{m_3}$, which leaves enough space for mode $m_3$ in the tail section. Let $v^*$ be the value of the items in this knapsack and $o^*$ denotes the optimal cost. Then

$$0 \leq v^* + \pi_d(m_1) + \pi_d(m_2) + \Delta\pi_{m_3} - o^* \leq \epsilon$$

    The value $v^*$ contains the leaps part as well as the fractions of the modes $m_1$ and $m_2$ (with continuous cost only).
  - Let $V_\Sigma$ be the value of all items in the multiset. For any solution to the knapsack problem with value $V$ we get a schedule $\sigma'$ with cost $\leq V_\Sigma - V + \pi_d(m_1) + \pi_d(m_2) + \Delta\pi_{m_3}$.

- The last case we will look at is empty pattern for both the head and tail sections. The schedule contains only leaps.

  For each type of leaps, $(m, m') \in M^+ \times M^-$, we build the following items for this knapsack problem instance: $\{(2^i \cdot \Delta t_{m,m'}, 2^i \cdot \Delta\pi_{m,m'}) \mid i \in \mathbb{N} \wedge 2^i \cdot \Delta\pi_{m,m'} \leq c^* \wedge 2^i \cdot \Delta t_{m,m'} \leq t_{\max}\}$.

  The produced 0-1 knapsack problem has the following properties:

  - $n$ leaps of of type $(m, m')$ in $\sigma$ can be achieved by picking the items for this type and corresponding to the binary representation of $n$; and
  - The volume of these items is $\geq t_{\max}$. Let $v^*$ be the value of the items in this knapsack and $o^*$ denotes the optimal cost. Then

$$0 \leq v^* - o^* \leq \epsilon$$

  - Let $V_\Sigma$ be the value of all items in the multiset. For any solution to the knapsack problem with value $V$ we get a schedule $\sigma'$ with cost $\leq V_\Sigma - V$.

## 4.6 Conclusions

We have studied cost optimisation in the general case of multi-mode single-dimension systems with discrete cost. We have identified this class as a class that arises naturally when studying the optimal control of heating or cooling systems with heaters and air-conditioners while the *idle* model can not be used to cool the room down. There is only one continuous variable (the temperature in our setting) in addition

to the time. We showed that the decision problem NP-complete. We studied the optimal schedules and showed that the optimal schedule can take any form out of 44 cases and it can be divided into thee sections: head, leaps and tail sections. Based on that, we proposed a three-approximation algorithm as well as an FPTAS approximation algorithm. The three-approximation has a $\mathcal{O}(|\mathcal{A}|^7)$ running time by choosing the same leap in the leap section of the optimal schedule form. The FPTAS approximation tries different 44 cases (we presented only 5 cases while the others are similar) that the schedule could take and picks the cheapest among them.

Summing up, we have identified a simple subclass of linear hybrid automata with an easy (LogSpace) optimal control problem over an infinite time horizon, and an optimal control problem over a finite time horizon, which is fast to approximate (FPTAS).

# Chapter 5

# Optimisation in Multiple Dimensional Multi-mode Systems

## 5.1 Introduction

We studied in Chapter 3 and Chapter 4 the optimisation of the multi-mode system in a one dimensional systems with and without a simplifying assumption, respectively. This chapter introduces the optimisation problem in multiple dimensional multi-mode systems with discrete costs. We start by showing that the optimal solution may not exist by introducing an example of controlling the temperature inside two rooms in which the optimal schedule can not be determined. So, we change our focus to the approximation algorithms. We start by studying the limit-safe and $\epsilon$-safe finite control. We show that finding an optimal limit-safe abstract schedule in $\mathcal{A}$ can be done in nondeterministic exponential time. We also show that if a limit-safe abstract schedule exists, then finding an $\epsilon$-safe $\epsilon$-optimal schedule can be done in deterministic polynomial space.

The following example shows that there may not be an optimal schedule for multiple dimensional multi-mode systems with a finite time horizon.

**Example 5.1.** *Consider a multi-mode system with three modes shown in Figure 5.1: $m_1, m_2, m_3$. The slope vectors in these modes are $A(m_1) = (1, 1)$, $A(m_2) = (1, -1)$ and $A(m_3) = (-1, 1)$, respectively. The continuous cost of using $m_1$ is $\pi_c(m_1) = 1$ and all the other costs are $0$.*

*Let $V_0 = V_{min} = \mathbb{0}_2$ and $V_{max} = \mathbb{1}_2$. Notice that we can only use $m_2$ or $m_3$ once we get out of the initial corner $V_0$. This can only be done using $M_1$. Now let the time horizon be $t_{max}$. Note that the following schedule $\sigma_\epsilon = (m_1, \epsilon), \big((m_2, t), (m_3, t)\big)^l$, where $t' = t_{max} - \epsilon$, $l = \lceil t'/\epsilon \rceil$, and $t = t'/2l$, has time horizon $t_{max}$ and total cost*

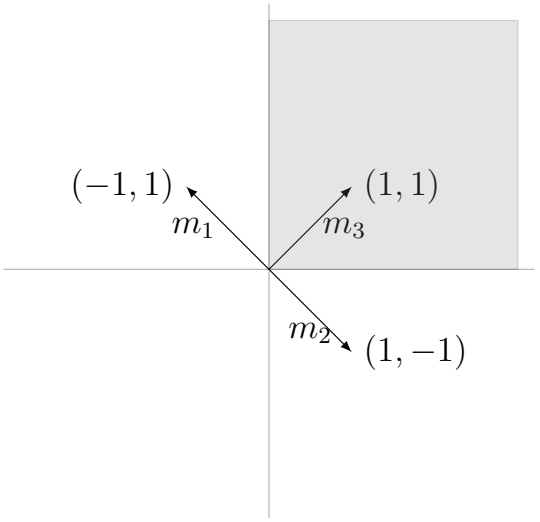



FIGURE 5.1: Example of MMS in multiple dimensions.

$\epsilon > 0$. *As* $\epsilon$ *can be made arbitrarily small but has to be* $> 0$, $\sigma_\epsilon$ *is an* $\epsilon$*-optimal schedule for all* $\epsilon > 0$*, but no optimal schedule exists.*

## 5.2 $\epsilon$-safe Schedules

The problem we face while searching for optimal schedules in multiple dimensional multi-mode systems is that the starting points may be located on the safety bounds and may only have modes like $m_2$ and $m_3$ in Example 5.1 using which we can not generate any safe schedule from the staring points located at the boundaries. So, we can overcome this problem by permitting a safety precision value $\epsilon$ and produce $\epsilon$*-safe* schedules.

**Definition 5.1.** An $\epsilon$*-safe* schedule with a safety bounds $V_{min}$ and $V_{max}$ is a schedule whose points $V(t)$ satisfy the condition $V_{min} - \epsilon \leq V(t) \leq V_{max} + \epsilon$, where $0 \leq t \leq t_{max}$ for finite schedules or $0 \leq t \leq \infty$ for infinite schedules.

We may also be interested in finding an approximation solutions with $\epsilon$ precision

**Definition 5.2.** An $\epsilon$*-safe* $\epsilon$*-optimal* schedule is a schedule that permits a safety deviation of $\epsilon$ and another cost deviation $\epsilon$ from the optimal schedule.

Note that in Example 5.1, for any $\epsilon > 0$, there exists an optimal $\epsilon$-safe schedule $\sigma$ with total cost 0: $\sigma_0 = \langle \big((m_2, t), (m_3, t)\big)^l \rangle$ where $l$ is defined as in Example 5.1. Our aim is to find an "abstract schedule" that, for any given $\epsilon > 0$, can be used to construct in polynomial time an $\epsilon$-safe $\epsilon$-optimal schedule.

Let $M^* = \{m \in M \mid \pi_d(m) = 0\}$ be the subset of modes without discrete costs. Note that, as shown in [9], the cost and safety of a schedule with $M^*$ modes only, depends only on the total amount of time spend in each of the $M^*$ modes. This motivates us to lump together any sequence of timed actions that only use $M^*$ modes and define an *abstract timed action (over $M^*$)* as a function $\mathbf{t} : M^* \to \mathbb{R}_{\geq 0}$.

**Definition 5.3.** A finite *abstract schedule* with time horizon $t_{\max}$ (of length $k$) is a finite sequence $\tau = \langle \mathbf{t}_1, (m_1, t_1), \mathbf{t}_2, (m_2, t_2), \ldots, (m_{k-1}, t_{k-1}), \mathbf{t}_k \rangle$ such that $\forall_i\, m_i \in M \setminus M^*$ and $\sum_{i \leq k, m \in M^*} \mathbf{t}_i(m) + \sum_{i<k} t_i = t_{\max}$.

The run of the abstract schedule $\tau$ is a sequence $\langle V_0, V_1, \ldots, V_{2k+1} \rangle$ such that, for all $i \leq k$, we have $V_{2i} = V_{2i-1} + A(m_i)t_i$ and $V_{2i+1} = V_{2i} + \sum_{m \in M^*} A(m)\mathbf{t}_i(m)$. We say that an abstract schedule is *limit-safe* if its run is safe. The total cost of an abstract schedule $\tau$ is defined as

$$\sum_{i \leq k, m \in M^*} \pi_c(m)\mathbf{t}_i(m) + \sum_{i<k} \left(\pi_d(m_i) + \pi_c(m_i)t_i\right).$$

Note that any safe schedule can be turned into a limit-safe abstract schedule with the same cost by simply replacing any maximal subsequence of consecutive timed actions that only use $M^*$ modes by a single abstract timed action. A limit-safe abstract schedule $\sigma$ is optimal if the total cost of all other limit-safe abstract schedules is higher than $\pi(\sigma)$. The following statement justifies the name "limit-safe".

**Proposition 5.4.** *Given a limit-safe abstract schedule $\tau$ and $\epsilon > 0$, we can construct in polynomial time an $\epsilon$-safe schedule $\sigma$ such that $\pi(\tau) = \pi(\sigma)$.*

*Proof.* Let $M^* = \{m_1, m_2, \ldots, m_j\}$. To obtain $\sigma$ from $\tau$, we replace each abstract timed action $\left\{\left(m, t_m\right) \mid m \in M^*\right\}$ by a sequence $\left((m_1, t_{m_1}/l), \ldots, (m_j, t_{m_j}/l)\right)^l$ for a sufficiently large $l \in \mathbb{N}$.

Sufficiently large means that, for $t^* = \sum_{m \in M^*} t_m$, $l > t^* \cdot \max_{m \in M^*} \|A(m)\|/\epsilon$. This choice guarantees that $\sum_{m \in M^*} \|A(m)\| \cdot t_m/l < \varepsilon$. Thus, when the abstract action $\left\{\left(m, t_m\right) \mid m \in M^*\right\}$ joins two states $V_{2i}, V_{2i+1}$ along the run $\langle V_0, V_1, \ldots, \ldots, V_{2k+1} \rangle$ of $\tau$, we know that this concrete schedule will cover the $l$-th part of $V_{2i}, V_{2i+1}$ after every sequence $(m_1, t_{m_1}/l), (m_2, t_{m_2}/l), \ldots, (m_j, t_{m_j}/l)$. As the safe set is convex, the start and end points of this sequence are safe points. Also, $\sum_{m \in M^*} \|A(m)\| \cdot t_m/l < \varepsilon$ implies that the points in the middle are $\epsilon$-safe. □

**Example 5.1 continues.** *An example limit-safe abstract schedule of length* 1 *is $\tau = \{(m_1, t_{max}/2), (m_2, t_{max}/2)\}$. Based on $\tau$ we can construct an $\epsilon$-safe schedule $\langle\left((m_1, t_{max}/2l), (m_2, t_{max}/2l)\right)^l\rangle$ where $l$ is any integer greater than $t_{max}/\epsilon$.*

We show that while looking for an ($\epsilon$-)safe ($\epsilon$-)optimal finite schedule, we can restrict our attention to angular schedules only where there are no two consecutive timed actions $(m_i, t_i), (m_{i+1}, t_{i+1})$ in $\sigma$ such that $A(m_i) = A(m_{i+1})$.

**Proposition 5.5.** *For every finite ($\epsilon$-)safe schedule with time horizon $t_{max}$ there exists an angular safe schedule with the same or lower cost.*

*Proof.* Let $\sigma$ be a finite safe schedule with two timed actions $(m_i, t_i), (m_{i+1}, t_{i+1})$ in $\sigma$ such that $A(m_i) = A(m_{i+1})$. (If no such timed actions exist then $\sigma$ is angular and we are done.) We can now replace these timed actions by a single timed action $(m, t_i + t_{i+1})$ such that $m$ is the mode from $m_i$ or $m_{i+1}$ with the lower continuous cost, and $m'$ the other mode. (I.e. $\{m, m'\} = \{m_i, m_{i+1}\}$ and $\pi_c(m) \leq \pi_c(m')$) For the resulting safe schedule $\sigma'$, it now holds that $\pi(\sigma') \leq \pi(\sigma) - \pi_d(m')$. □

Henceforth, we assume that all finite schedules are angular.

## 5.3 Optimisation of Multiple dimensional Multi-mode Systems without Discrete Costs

This section presents a simple case of the optimisation in the multiple dimensional multi-mode system $\mathcal{A}$ with a finite time horizon $t_{max}$ where all the discrete costs for all the modes equal to zero $\forall_{m \in M} \pi_d(m) = 0$. As shown in Algorithm 7, we study the safe reachability problem of the multiple dimensional multi-mode system with finite time horizon $t_{max}$ and return a Yes/No answer if a safe schedule exists. This algorithm is based on an adaptation of [9, Algorithm 2]. We assume that the starting points $V_0$ belong to the interior safe set. If there exists a safe schedule, the algorithm returns a safe schedule with the minimum continuous cost.

## 5.4 Complexity of Limit-safe and $\epsilon$-safe Finite Control

In the rest of this section we fix a (multi-dimensional) multi-mode system $\mathcal{A}$ and time horizon $t_{\max}$.

**Theorem 5.6.** *If a limit-safe abstract schedule exists in $\mathcal{A}$, then there exists one of exponential length and its symbolic representation can be constructed in polynomial time.*

**Algorithm 7** An algorithm checking whether any safe schedule exists and if so finding one with the minimal total continuous-cost.

**Input:** MMS $\mathcal{A} = (M = \{m_1, \ldots, m_k\}, N, A, \pi_c, \pi_d \equiv 0, V_{\min}, V_{\max}, V_0)$, target point $V_{\text{end}}$ and $t > 0$ such that all modes of $\mathcal{A}$ are safe at $V_0$ and $V_{\text{end}}$ for time $t$.

**Output:** NO, if no safe schedule from $V_0$ to $V_{\text{end}}$ exists, and a continuos-cost-optimal schedule (of at most exponential length), otherwise.

1: Check whether the following linear programming problem with variables $\{t^{(m)}\}_{m \in M}$ has a solution.

$$\text{Minimise} \sum_{m \in M} \pi_c(m) t^{(m)} \text{ subject to:}$$
$$V_0 + \sum_{m \in M} A(m) t^{(m)} = V_{\text{end}} \text{ and}$$
$$t^{(m)} \geq 0 \text{ for all } m \in M.$$

2: **if** no satisfying assignment exists **then**
3: **return** NO
4: **else**
5: Find a polynomial sized assignment $\{t^{(m)}\}_{m \in M}$.
6: Let $l$ be the smallest natural number greater or equal to $\sum_{m \in M} t^{(m)}/t$. (Note that this number is at most exponential in the size of the input and can be written down using polynomially many bits. Also, $t$ is the amount of time where all the modes of $\mathcal{A}$ are safe.)
7: **return** the schedule $\big((m_1, t^{(m_1)}/l), (m_2, t^{(m_2)}/l), \ldots, (m_k, t^{(m_k)}/l)\big)^l$.
8: **end if**

*sketch.* Before we formally prove this theorem, we need to introduce first a bit of terminology. We call a mode $m$ *safe for time* $t > 0$ at $V \in S := \{x \in \mathbb{R}^N : V_{\min} \leq x \leq V_{\max}\}$ if $V + A(m)t \in S$. Also, $m$ is *safe* at $V$ if there exists $t > 0$ such that $m$ is safe for time $t$ at $V$. We say that a *coordinate of a state*, $V \in S$, *is at the border* if that coordinate in $V$ is equal to the corresponding coordinate in $V_{\min}$ or $V_{\max}$.

Our algorithm first removes from $M$ all modes that will never be safe to use in a limit-safe schedule. This procedure can be found between lines 1 – 8 of Algorithm 8. This is an adaptation of [9, Theorem 7] where an algorithm was given for finding safe modes that can ever be used in a schedule with infinite time horizon. The main difference here is that the modes in $M^*$ can always be used in a limit-safe abstract schedule even if they are not safe to use. We find here a sequence of sets of modes $M^* = M_0 \subset M_1 \subset M_2 \subset \ldots$ such $M_{i+1}$ is the set of modes that are safe at a state reachable from $V_0$ via a limit-safe abstract schedule that only uses modes from $M_i$. Note that at some step $k \leq |M|$ this sequence will stabilise, i.e.

$M_k = M_{k+1}$. Similarly as in the proof of [9, Theorem 7], we can show that no mode from $M \setminus M_k$ can ever be used by a limit-safe abstract schedule. As a result, we can remove all these modes from $M$.

Next, we remove all modes that cannot be part of a limit-safe abstract schedule with time horizon $t_{\max}$. For this, for each $m$, we formulate a very similar linear programme (LP) as above (cf. lines 9 – 11 of Algorithm 8) where we ask for the time delay of $m$ to be positive and the total time delay of all the modes to be $t_{\max}$. By a simple adaptation of the proof of [9, Theorem 4], if this LP is not satisfiable then $m$ can be removed from $\mathcal{A}$.

Next, we look for the easiest possible target state $V_{\text{end}}$ that can potentially be reached using a limit-safe abstract schedule from $V_0$ with time horizon $t_{\max}$. For this, $V_{\text{end}}$ has to have the least number of coordinates at the border of the safe set. Note that this is well-defined, because if $V$ and $V'$ are two points reachable from $V_0$ via a limit-safe abstract schedules $\tau$ and $\tau'$ with time horizon $t_{\max}$, respectively, then $\tau/2$ (i.e. divide all abstract and timed actions delays in $\tau$ by 2) followed by $\tau'/2$, is also a limit-safe abstract schedule with time horizon $t_{\max}$, which reaches $(V + V')/2$. However, $(V + V')/2$ has a coordinate at the border iff both $V$ and $V'$ have it as well. This shows that there is a state with a minimum number of coordinates at the border.

To find the coordinates that need to be at the border we will use the following LP. We have a variable $x_i$ for each dimension $i \leq N$ and a constraint that requires $x_i$ to be less or equal to the $i$-th coordinate of $V_{\max} - V_{\text{end}}$ **and** $V_{\text{end}} - V_{\min}$. We also add that $\sum_{m \in M} t_m = t_{\max}$ and $V_{\text{end}} = V_0 + \sum_{m \in M} t_m \cdot A(m)$, with the objective *Maximise* $\sum_i x_i$. If the value of the objective is $> 0$, we will get to know a new coordinate that does not have to be at the border. We then remove it from the LP and run the LP again. Once the objective is 0, then all the remaining coordinates, $I$, have to be at the border and the solution to this LP tells us, at which border the solution has to be located (it cannot possibly be at the border of both $V_{\min}$ and $V_{\max}$ as then we could reach the middle).

Next, in order to bound the length of a limit-safe abstract schedule by an exponential in the size of the input, we not only need a state with the minimum number of coordinates at the border, but also sufficiently far way from the border. Otherwise, we may need super-exponentially (i.e. it means that we will need a large number of consecutive timed-actions with small time periods) many timed actions to reach it. In order to find such a point, we replace all $x_i$-s in the previously defined LP by a single variable $x$ which is smaller or equal to all the coordinates of $V_{\max} - V_{\text{end}}$ and $V_{\text{end}} - V_{\min}$ from $I$. We then set the objective to *Maximise* $x$, which will give us a suitable easy target state $V_{\text{end}}$.

Now, consider $\mathcal{A}'$, which is the same as $\mathcal{A}$ but with all slopes negated (i.e. $A'(m) = -A(m)$ for all $m \in M$). We claim that $V_{\text{end}}$ is reachable from $V_0$ using a limit-safe abstract schedule $\tau$ iff $(V_0 + V_{\text{end}})/2$ is reachable from $V_0$ in $\mathcal{A}$ with time horizon $t_{\max}/2$ and $(V_0 + V_{\text{end}})/2$ is reachable from $V_{\text{end}}$ in $\mathcal{A}'$ with time horizon $t_{\max}/2$; this again follows by considering $\tau/2$. Note that a coordinate of $(V_0 + V_{\text{end}})/2$ is at the border iff it is at the border in both $V_0$ and $V_{\text{end}}$.

This way we reduced our problem to just checking whether a limit-safe abstract schedule exists from one point to another more permissive point (i.e. where the set of safe modes is at least as big) within a given time horizon. Algorithm 8 solves this problem and constructs (if there exists one) a limit-safe abstract schedule of at most exponential length with these properties. It again reuses the same constructions as above, e.g. constructs exactly the same sequence of sets of modes $M^* = M_0 \subset M_1 \subset \ldots \subset M_k$. Its correctness follows by a similar reasoning as above. We now need to invoke this algorithm twice: to check that $(V_0 + V_{\text{end}})/2$ is reachable from $V_0$ with time horizon $t_{\max}/2$ and that $(V_0 + V_{\text{end}})/2$ is reachable from $V_{\text{end}}$ with time horizon $t_{\max}/2$ in $\mathcal{A}'$. If at least one of these calls return NO, then no limit-safe abstract schedule from $V_0$ to $V_{\text{end}}$ can exist. Otherwise, let $\sigma$ and $\sigma'$ be the schedules returned by these two calls, respectively. Then the concatenation of $\sigma$ with the reverse of $\sigma'$ is a limit-safe abstract schedule that reaches $V_{\text{end}}$ from $V_0$ with time horizon $t_{\max}$.

□

**Theorem 5.7.** *Finding an optimal limit-safe abstract schedule in $\mathcal{A}$ can be done in nondeterministic exponential time.*

*Proof.* The limit-safe abstract schedule constructed in Theorem 5.6 has exponential length. To establish a nondeterministic exponential upper bound, we can guess the modes (and the order in which they occur). With them, we can produce an exponentially sized linear program, which encodes that the run of the abstract schedule is safe and minimises the total cost incurred. □

Theorem 5.7 and Proposition 5.4 immediately give us the following.

**Corollary 5.8.** *If a limit-safe abstract schedule exists in $\mathcal{A}$, then for any $\epsilon > 0$ an $\epsilon$-safe schedule with the same cost can be found in nondeterministic exponential time.*

Moreover, from Theorem 5.6 and the fact that in the case of multi-mode systems with no discrete costs all abstract schedules have length 1, we get the following.

**Corollary 5.9.** *Finding an optimal limit-safe abstract schedule for multi-mode systems with no discrete costs can be done in polynomial time.*

We can reduce the computational complexity in the general model if we are willing to sacrifice optimality for $\epsilon$-optimality.

**Theorem 5.10.** *If a limit-safe abstract schedule exists, then finding an $\epsilon$-safe $\epsilon$-optimal schedule can be done in deterministic polynomial space.*

*Proof.* When reconsidering the linear programme from the end of the proof of Theorem 5.7, we can guess the intermediate states in polynomial space (and thus guess and output the schedule) as long as all states along the run (including the time that passed so far) are representable in polynomial space.

Otherwise we use the opportunity to deviate by up to $\epsilon$ from the safe set by increasing or decreasing the duration of each timed action up to some $\delta > 0$, in order to keep the intermediate values representable in space polynomial in $|\mathcal{A}|$ and $\epsilon$. However, we apply these changes in a way that the overall time remains $t_{\max}$. Clearly this is possible, because within $\delta/2$ of the actual time point of each state along the run, there is a value whose number of digits in the standard decimal notation is at most equal to the sum of the number of digits in $\delta/2$ and $t_{\max}$. Picking any such point for every interval would induce a schedule with the required property and they can be simply guessed one by one.

The final imprecision introduced by this operation is at most $b{\cdot}\delta{\cdot}\max_{m\in M}|A(m)|$, where $b$ is a bound on the number of timed actions in a limit-safe schedule, which is exponential in $|\mathcal{A}|$. If we choose $\delta = \epsilon/(b \cdot \max_{m\in M}|A(m)|)$, then we will get the required precision.

Although our algorithm is nondeterministic, due to Savitch's theorem, it can be implemented in deterministic polynomial space. □

## 5.5 Approximation Algorithms for the Multiple Dimensional Multi-mode Systems with Discrete Costs

The first idea of how we can produce approximation schedules is to limit the number of switches from paying only continuous costs to paying discrete costs as well. The approximate schedules we produce limit the number of timed actions for the modes that have discrete costs. We assume that the approximate schedules can use as any number of the timed actions for the modes that do not have discrete costs.

**Algorithm 8** Finding a limit-safe schedule to target state $V_{\text{end}}$ with time horizon $t_{\max}$.

**Input:** Multi-mode system $\mathcal{A} = (M, N, A, \pi_c, \pi_d, V_{\min}, V_{\max}, V_0)$, set of modes $M^*$ with zero discrete costs, time horizon $t_{\max}$, and target state $V_{\text{end}}$ such that any mode safe at $V_0$ is safe as $V_{\text{end}}$.

**Output:** NO if no safe schedule with time horizon $t_{\max}$ exists from $V_0$ to $V_{\text{end}}$, and such a schedule, otherwise.

1: $k := 0; M_0 := M^*$;
2: **repeat**
3: **for each** mode $q \in M \setminus M_{k-1}$ **do**
4: **if** the following set of linear constraints is satisfiable for some assignment to the variables $t, \{t_0^{(m)}\}_{m\in M_0}, \{t_1^{(m)}\}_{m\in M_1}, \ldots, \{t_{k-1}^{(m)}\}_{m\in M_{k-1}}$:

$$
\begin{aligned}
&\cdot\ t > 0 \\
&\quad \textit{For all } i = 0, \ldots, k-1: \\
&\qquad \cdot\ t_i^{(m)} \geq 0 \ \textit{for all } m \in M_i \\
&\qquad \cdot\ V_{i+1} = V_i + \sum_{m\in M_i} A(m) t_i^{(m)} \\
&\qquad \cdot\ V_{\min} \leq V_{i+1} \leq V_{\max} \\
&\cdot\ V_{\min} \leq V_k + A(q)t \leq V_{\max}
\end{aligned}
\tag{5.1}
$$

**then**
5: $M_k := M_{k-1} \cup \{q\}$;
6: **end if**
7: **end for**
8: **until** $M_k = M_{k-1}$

As we introduced before in Definition 5.3, the set of consecutive timed actions for the modes without discrete costs produces an abstract schedule part. We enable using abstract schedule parts where every one is followed by only one timed action that uses a mode with discrete cost. As shown in Algorithm 9, we specify the maximum number of allowed switches as $s_{max}$. We use the concept of bounded unfolding technique [40] to find an approximate solution that uses modes with discrete costs at most $s_{max}$ number of times. If the multi-mode system $\mathcal{A}$ does not have any mode $m \in M$ where $\pi_d(m) > 0$, so the algorithm finds the minimal $\epsilon$-safe schedule (lines 1-3). If the multi-mode system $\mathcal{A}$ has at least only one mode with discrete cost, lines (4-13) find the best approximation schedule with a maximum discrete switches of $s_{max}$ and output it. So, what if the optimal schedule contains too many timed-actions with discrete costs. The approximation solution generated from Algorithm 9 may be too far from the optimal one. However Algorithm 9 takes a long time to run as it test all the possibilities of using modes with discrete costs when enabling a new switch from paying only continuous cost (when using

9: $k := k - 1$;
10: **for each** $j = 0, \ldots, k$ **and** $q \in M_j$ **do**
11: **if** the following set of linear constraints is **not** satisfiable for any assignment to the variables $t, \{t_0^{(m)}\}_{m \in M_0}, \{t_1^{(m)}\}_{m \in M_1}, \ldots, \{t_k^{(m)}\}_{m \in M_k}$:

- $t_j^{(q)} > 0$
- *For all* $i = 0, \ldots, k-1$ :
  - $t_i^{(m)} \geq 0$ *for all* $m \in M_i$
  - $V_{i+1} = V_i + \sum_{m \in M_i} A(m) t_i^{(m)}$
  - $V_{\min} \leq V_{i+1} \leq V_{\max}$
- $\sum_{i=0}^{k} \sum_{m \in M_i} t_i^{(m)} = t_{\max}$

**then**
12: $M_j := M_j \setminus \{q\}$;
13: **end if**
14: **if** the following set of linear constraints is **not** satisfiable for any assignment to the variables $\{t_0^{(m)}\}_{m \in M_0}, \{t_1^{(m)}\}_{m \in M_1}, \ldots, \{t_k^{(m)}\}_{m \in M_k}$:

- *For all* $i = 0, \ldots, k$ :
  - $t_i^{(m)} > 0$ *for all* $m \in M_i$
  - $V_{i+1} = V_i + \sum_{m \in M_i} A(m) t_i^{(m)}$
  - $V_{\min} \leq V_{i+1} \leq V_{\max}$
- $\sum_{i=0}^{k} \sum_{m \in M_i} t_i^{(m)} = t_{\max}$

**then**
15: **return** NO
16: **end if**
17: **end for**
18: Compute a polynomially sized solution to the linear program in step 14 and use it in the next line.
19: **return** the schedule created by composing the following schedules obtained by repeatedly calling [9, Algorithm 2] to find a safe schedule:

- from $V_0$ to $V_1$ using only modes in $M_0$ with the safe time bound $t = \min_{m \in M_0} t_0^{(m)}$,
- from $V_1$ to $V_2$ using only modes in $M_1$ with the safe time bound $t = \min_{m \in M_1} t_1^{(m)}$,
- from $V_k$ to $V_{k+1}$ using only modes in $M_k$ with the safe time bound $t = \min_{m \in M_k} t_k^{(m)}$.

the modes that do not have discrete costs) to discrete cost as well (when using a mode that has a discrete cost), the approximation solutions generated are not always good approximations.

Algorithm 10 overcomes the previous problem by using the idea behind the greedy unfolding approximation technique. The idea is simply like finding the best mode with discrete cost –that is used after the abstract schedule part– which results in minimal average cost and fix it. So, when we add new switch (from continuous paying to discrete paying) we use the same fixed modes with discrete costs that belong to the old switches beside the new mode with discrete cost that belongs to the new switch. So, every new switch the algorithm solves only $|M^*|$ linear programming problem instead of $|M^*|^s$ problems –where $s$ is the switch number– which improves the algorithm running time and gives us the opportunity to discover more switches which results in finding a better approximation solution than the one obtained from Algorithm 9. So, the algorithm keeps adding more discrete-cost switches until the difference between the minimal costs produced from the last two switches is smaller than $\rho o^*$ where $\rho$ is the approximation precision and $o^*$ is the best approximation using only one discrete-cost switch with is an upper bound of the optimal solution.

## 5.6 Conclusions

For the multiple dimensional multi-mode system, the optimal safe schedule may not exist. This problem may occur if the initial point lies on the safe boundaries and there is no mode that can be used to get rid of the boundary and preserve the safety constrains. A solution for this is to enable $\epsilon$-safety deviation where the safety boundaries are extended from $(V_{min}, V_{max})$ to $((V_{min} - \epsilon), (V_{max} + \epsilon))$. We proved that if a limit-safe abstract schedule exists in $\mathcal{A}$, then there exists one of exponential length and it can be constructed in polynomial time. We also showed that finding an optimal limit-safe abstract schedule in $\mathcal{A}$ can be done in nondeterministic exponential time. Next, we showed that if a limit-safe abstract schedule exists, then finding an $\epsilon$-safe $\epsilon$-optimal schedule can be done in deterministic polynomial space and proposed an algorithm to find it.

**Algorithm 9** Finding $\epsilon$-safe bounded unfolding approximation finite schedule with time horizon $t_{\max}$.

**Input:** Multi-mode system $\mathcal{A} = (M, N, A, \pi_c, \pi_d, V_{\min}, V_{\max}, V_0)$, set of modes $M^*$ with zero discrete costs, time horizon $t_{\max}$, safety precision parameter $\epsilon$, maximum number of switches $s_{max}$, time $t > 0$ where all the modes $m \in M \setminus M*$ are safe.

**Output:** The approximation schedule $\sigma$.

1: **if** $(|M \backslash M^*| = 0)$ **then**

2:

$$\text{Minimise} \sum_{m \in M} \pi_c(m) t_m \text{ subject to:}$$

$$V_{min} - \epsilon \leq V_0 + \sum_{m \in M} A(m) t_m \leq V_{max} + \epsilon$$

$$\sum_{m \in M} t_m \geq t_{max}$$

$$t_m \geq 0 \text{ for all } m \in M.$$

3:

- Find a polynomial sized assignment $\{t_m\}_{m \in M}$.
- Let $l$ be the smallest natural number greater or equal to $\sum_{m \in M} t_m / t$. (Note that this number is at most exponential in the size of the input and can be written down using polynomially many bits.)
- **return** the schedule $\sigma = \big((m_1, t_{m_1}/l), (m_2, t_{m_2}/l), \ldots, (m_k, t_{m_k}/l)\big)^l$.

4: **else**

5: **for each** $i \in \{1, \ldots, s_{max}\}$ **do**

6:

$$\forall_{j \leq i}, m_j^* \in M^* \text{ and } m \in M \setminus M^*$$

$$\text{Minimise} \sum_{j \leq i} \sum_{m \in M \setminus M^*} (\pi_c(m) t_m^j + \pi_d(m_j^*) + \pi_c(m_j^*) t_{m_j^*}^j), \text{ where } m_j^* \in M^*$$

subject to:

$$V_{min} - \epsilon \leq V_0 + \sum_{j<i} \sum_{m \in M \setminus M^*} (A(m) t_m^j + A(m_j^*) t_{m_j^*}^j) +$$

$$\sum_{m \in M \setminus M^*; j:=i} A(m) t_m^j \leq V_{max} + \epsilon$$

$$V_{min} - \epsilon \leq V_0 + \sum_{j \leq i} \sum_{m \in M \setminus M^*} (A(m) t_m^j + A(m_j^*) t_{m_j^*}^j) \leq V_{max} + \epsilon$$

$$\sum_{j \leq i} \sum_{m \in M \setminus M^*} (t_m^j + t_{m_j^*}^j) \geq t_{max}$$

$$t_m^j \geq 0, t_{m_j^*}^j \geq 0 \text{ for all } m \in M \setminus M^* \text{ and } m_j^* \in M^* \text{ and } j \leq i.$$

7: **end for**

8: Save the values $(t_m^j, t_{m_j^*}^j, m_j^*)$ that result in the minimal total cost among all the values generated from lines 5–7 and save its value of $i^* := i$;

9: **for each** $j \leq i^*$ **do**
10: Find

- A polynomially sized assignment $\{t^j_m\}_{m \in M \setminus M^*}$.
- Let $l_j$ be the smallest natural number greater or equal to $\sum_{m \in M \setminus M^*} t^j_m / t$.
- Add to the output schedule $\sigma$ the part $\left(((m_1, t^j_{m_1}/l_j), (m_2, t^j_{m_2}/l_j), \ldots, (m_k, t^j_{m_k}/l_j)\right)^l, (m^*_j, t^j_{m^*_j}))$.

11: **end for**
12: **return** the schedule $\sigma$.
13: **end if**

**Algorithm 10** Finding greedy unfolding approximation $\epsilon$-safe finite schedule with time horizon $t_{\max}$.

**Input:** Multi-mode system $\mathcal{A} = (M, N, A, \pi_c, \pi_d, V_{\min}, V_{\max}, V_0)$, set of modes $M^*$ with zero discrete costs, time horizon $t_{\max}$, safety precision parameter $\epsilon$, approximation precision $\rho$, time $t > 0$ where all the modes $m \in M \setminus M*$ are safe.

**Output:** The approximation schedule $\sigma$.

1: Initialise $k := 1$; $\pi_{new} := \infty$; $\pi_{old} := \infty$; $o^* = \infty$

2: **if** $(|M \backslash M^*| = 0)$ **then**

3:

$$\begin{aligned} \text{Minimise} \sum_{m \in M} \pi_c(m) t_m \text{ subject to:} \\ V_{min} - \epsilon \leq V_0 + \sum_{m \in M} A(m) t_m \leq V_{max} + \epsilon \\ \sum_{m \in M} t_m \geq t_{max} \\ t_m \geq \ 0 \text{ for all } m \in M. \end{aligned}$$

4:

- Find a polynomial sized assignment $\{t_m\}_{m \in M}$.
- Let $l$ be the smallest natural number greater or equal to $\sum_{m \in M} t_m / t$. (Note that this number is at most exponential in the size of the input and can be written down using polynomially many bits.)
- **return** the schedule $\sigma = \big((m_1, t_{m_1}/l), (m_2, t_{m_2}/l), \ldots, (m_k, t_{m_k}/l)\big)^l$.

5: **else**

6: Consider adding the first timed action with zero discrete cost. Solve these $|M^*|$ linear programming problem.

$$\text{Minimise} \sum_{m \in M \setminus M^*} \pi_c(m) t^1_m + \pi_d(m^*) + \pi_c(m^*) t^1_{m^*}, \text{ where } m^* \in M^*$$

subject to:

$$\begin{aligned} V_{min} - \epsilon \leq V_0 + \sum_{m \in M \setminus M^*} (A(m) t^1_m + A(m^*) t^1_{m^*}) \leq V_{max} + \epsilon \\ \sum_{m \in M \setminus M^*} t^1_m + t^1_{m^*} \geq t_{max} \\ t^1_m \geq \ 0 \text{ for all } m \in M \setminus M^* \\ t^1_{m^*} \geq \ 0 \text{ for all } m^* \in M^*. \end{aligned}$$

7: Find the schedule with the minimal total cost from the $|M^*|$ linear programming problems solved in step 6. Find the mode with discrete mode $m^*$ that used with that minimal schedule and assign it to $m^*_k := m^*$ where $(k = 1)$. Also, assign the minimal cost value to the variables $\pi_{new}$ and $o^*$.

8: **do**

9: $k := k + 1$

10: Add one more timed action with discrete cost and solve the next $|M^*|$ linear programming problems.

$$\forall_{j<k}, m_j^* \text{ is pre-setted, } m_k^* \in M^* \text{ and } m \in M \setminus M^*$$

$$\text{Minimise} \sum_{j \le k} \sum_{m \in M \setminus M^*} (\pi_c(m) t_m^j + \pi_d(m_j^*) + \pi_c(m_j^*) t_{m_j^*}^j), \text{ where } m_k^* \in M^*$$

subject to:

$$V_{min} - \epsilon \le V_0 + \sum_{j<k} \sum_{m \in M \setminus M^*} (A(m) t_m^j + A(m_j^*) t_{m_j^*}^j) +$$

$$\sum_{m \in M \setminus M^*} A(m) t_m^k \le V_{max} + \epsilon$$

$$V_{min} - \epsilon \le V_0 + \sum_{j \le k} \sum_{m \in M \setminus M^*} (A(m) t_m^j + A(m_j^*) t_{m_j^*}^j) \le V_{max} + \epsilon$$

$$\sum_{j \le k} \sum_{m \in M \setminus M^*} (t_m^j + t_{m_j^*}^j) \ge t_{max}$$

$$t_m^j \ge 0 \text{ and } t_{m_j^*}^j \ge 0 \text{ for all } m \in M \setminus M^*, m_k^* \in M^*.$$

11: After solving the $|M^*|$ integer programming problem considered in line 10 do.

- Find the schedule $\sigma'$ with the minimal total cost and save the mode with discrete cost $m_k^*$ that was added recently. Also save the times $t_m^j$ and $t_{m_j^*}^j$ where $0 \le j \le k$.
- Do the following updates
  - $\pi_{old} := \pi_{new}$;
  - $\pi_{new} = \pi(\sigma')$;

12: **while** $(\pi_{old} - \pi_{new}) > \rho \times o^*$

13: **for each** $j \le k$ **do**

14: Find

- A polynomially sized assignment $\{t_m^j\}_{m \in M \setminus M^*}$.
- Find $m_j^*$.
- Let $l_j$ be the smallest natural number greater or equal to $\sum_{m \in M \setminus M^*} t_m^j / t$.
- Add to the output schedule $\sigma$ the part $\big(((m_1, t_{m_1}^j / l_j), (m_2, t_{m_2}^j / l_j), \ldots, (m_k, t_{m_k}^j / l_j))^l, (m^*, t_{m^*}^j)\big)$.

15: **end for**

16: **return** the schedule $\sigma$.

17: **end if**

# Chapter 6

# Experiments, Comparisons and Results

## 6.1 Introduction

In this chapter, we compare the performance of the several algorithms that we devised in Chapter 3 for the optimal finite time horizon control problem for multi-mode systems. Namely, we compare Algorithm 4 which is a constant factor approximation algorithm, Algorithm 3 for finding the optimal solution based on the integer programming formulation of our optimisation problem stated explicitly in this algorithm, and the FPTAS algorithm shown by Algorithm 6. All algorithms were implemented in Java and all tests were conducted on a Lenovo ideapad 110 with AMD A8-7410 APU 2.20 GHz and 8GB of RAM. For integer linear programming (ILP) we used the Gurobi Optimizer 7.0 with academic license.

We have conducted tests for two different categories of data instances. The first category is the normal data instances, where the instances' values generated with small values. The other type is for the hard data instances, which are randomly generated instances with various correlation characteristics between the coefficients as defined in Section 5.5 of [32] for the 0-1 knapsack problem. Such instances are some of the hardest to solve for most algorithms for the 0-1 knapsack problem.

## 6.2 Testing with Normal Instances

In this section, we tested our algorithms over weakly correlated instances with small coefficient values, where for all $i \in M$, we pick both $\Delta t_i$ and $\Delta \pi_i$ randomly from the interval $[1, R]$, where $R$ is some constant with a small value (we used $R = 100$). We also, run the test over a finite number of modes where $M =$

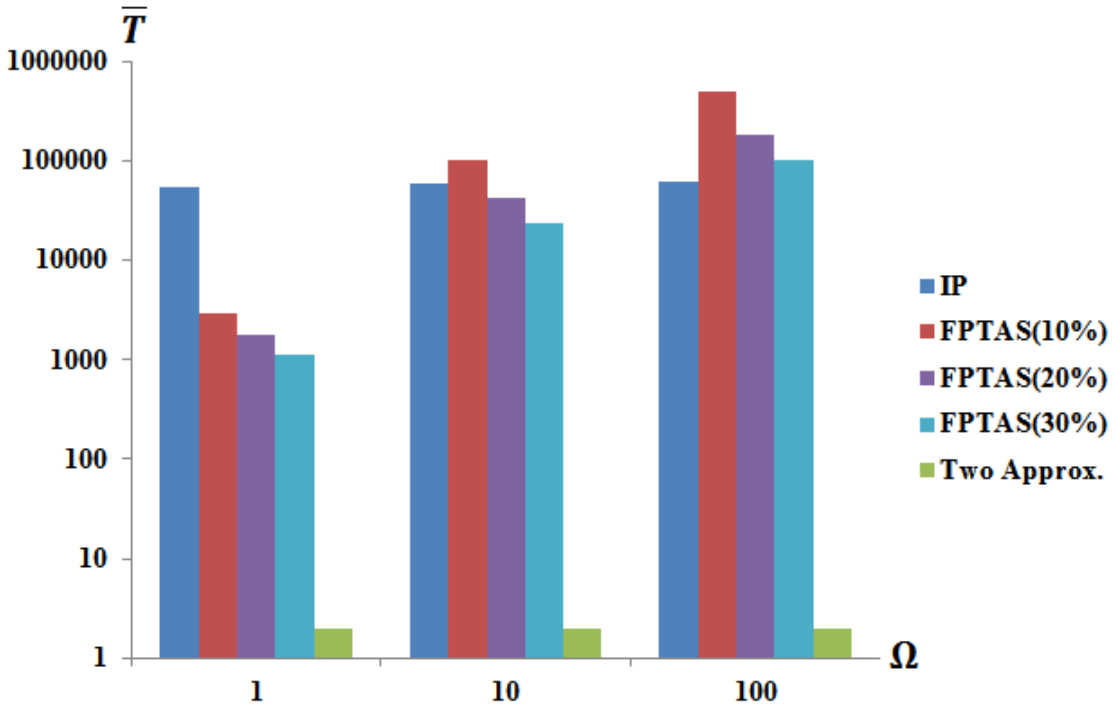



FIGURE 6.1: Average execution time in microsecond for the Integer programming algorithm, FPATS algorithm with $\epsilon = \{0.1, 0.2, 0.3\}$ and Two approximation algorithm for $t_{max,i} = \Omega \times \sum_{j \leq i} \Delta t_j$ and $\Omega \in \{1, 10, 100\}$.

$\{2, 4, 6, 8, 10\}$. For every mode $i \in M$, we generated 1000 knapsack problem with different values. For each knapsack problem, we run the test for three different knapsack volume values $t_{max}$. We refer to the volume for a knapsack problem with $i$ modes by $t_{max,i}$, where $i \in M$. We calculate $t_{max,i} = \Omega \cdot \sum_{j \leq i} \Delta t_j$ and $\Omega \in \{1, 10, 100\}$.

Figure 6.1 shows the average execution time in logarithmic scale with different $t_{max}$. The horizontal axis represents the $\Omega$ value. The vertical axis is a logarithmic scale for the average time value in microseconds. The execution time is obtained by $\overline{T}$ (measured in microsecond) and can be calculated as $\overline{T} = \frac{\sum_{0 \leq i < 1000, m \in M} t_{m,i}}{|M| \cdot 1000}$ where $t_{m,i}$ is the execution time for the test trial number $i$ and the number of modes is equal to $m$.

We can see that, for $\Omega$ with small value ($\Omega = 1$), the integer programming algorithm has the worst average execution time, while, for the other two values ($\Omega = 10$ and $\Omega = 100$), the integer programming algorithm time is not too far from the FPTAS algoithm execution time. It is unsurprising that the execution time for the two approximation algorithm is the minimal and it is constant (ie. it does not depend on the value of $\Omega$).

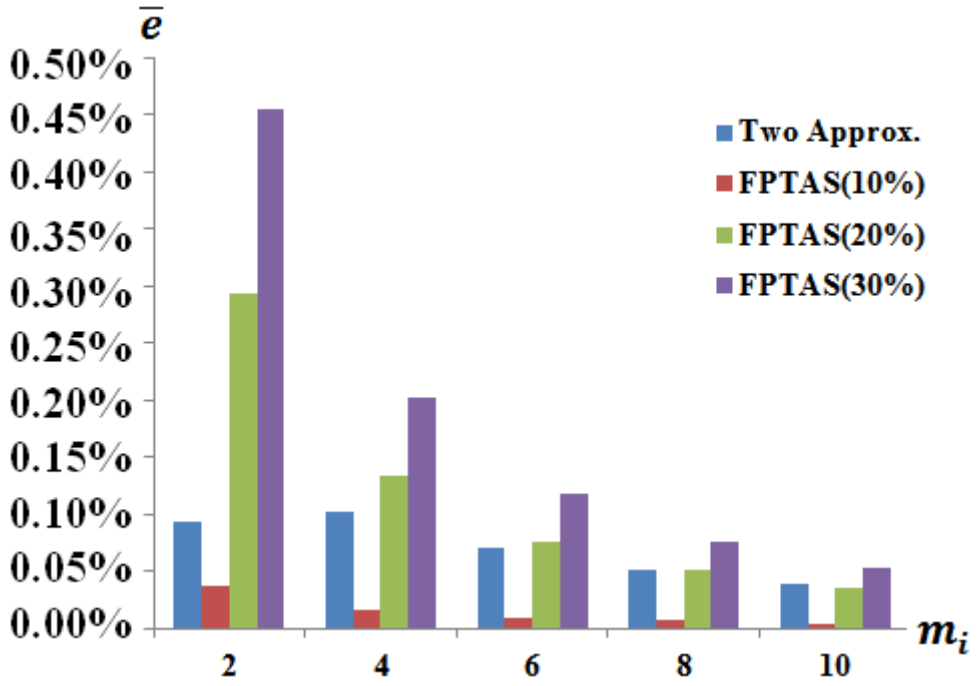



FIGURE 6.2: Average relative error for FPATS algorithm with $\epsilon = \{0.1, 0.2, 0.3\}$ and Two approximation algorithm for knapsack problems with modes $m_i \in \{2, 4, 6, 8, 10\}$ and various $t_{max,i} = \Omega \times \sum_{j \leq i} \Delta t_j$ where $\Omega \in \{1, 10, 100\}$.

Figures 6.2 and 6.3 show the average relative error and the maximum relative error ,respectively, for the normal instances test with modes $m_i = \{2, 4, 6, 8, 10\}$ of the FPTAS algorithm with $\epsilon \in \{0.1, 0.2, 0.3\}$ and the two approximation algorithm. The horizontal axis represents the number of modes for the knapsack problem (instances numbers) $m_i$, while the vertical axis represents the average percentage error $\overline{e}$ for Figure 6.2 and the maximum percentage error $e_{max}$ for Figure 6.3. The average error $\overline{e}$ is calculated by $\overline{e} = \sum_{1 \leq i \leq 1000, \Omega \in \{1,10,100\}} \frac{A-O}{O} * 100$, where $i$ represents the testing trial number, $O$ is the optimal answer generated by the integer programming algorithm and $A$ is the approximation answer generated from the approximation algorithms shown before with the same testing conditions. We can conclude that all the approximation algorithms have a very low average error but the problem with the two approximation algorithm can be shown in Figure 6.3 as the maximum relative error reached 20% and can reach 100% in some cases as shown in Section 3.5.

## 6.3 Testing with Hard Instances

In this section, we use strongly correlated, weakly correlated and uncorrelated hard instances. Instances of different correlation types are defined as follows.

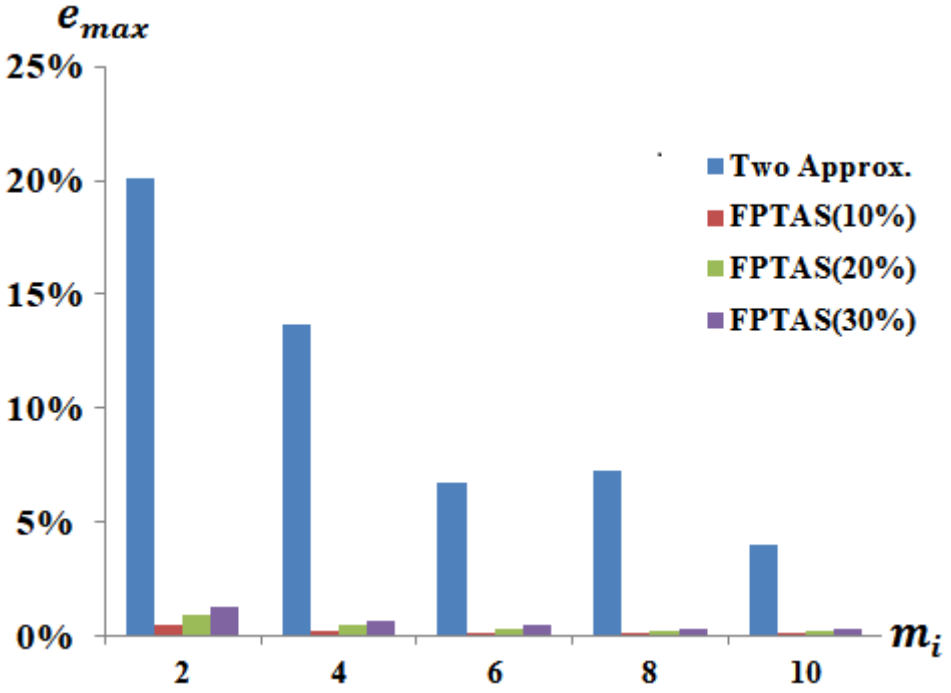



FIGURE 6.3: Maximum relative error for FPATS algorithm with $\epsilon = \{0.1, 0.2, 0.3\}$ and Two approximation algorithm for knapsack problems with modes $m_i \in \{2, 4, 6, 8, 10\}$ and various $t_{max,i} = \Omega \times \sum_{j \leq i} \Delta t_j$ where $\Omega \in \{1, 10, 100\}$.

1. *Strongly correlated instances.* For all $i \in M$, we pick $\Delta t_i$ uniformly at random from the interval $[1, R]$, where $R$ is some constant. We then assign $\Delta \pi_i = \Delta t_i + \frac{R}{10}$ for $i \in M$.

2. *Weakly correlated instances.* For all $i \in M$, we pick both $\Delta t_i$ and $\Delta \pi_i$ randomly from the interval $[1, R]$, where $R$ is some constant.

3. *Uncorrelated instances.* For all $i \in M$, we pick $\Delta t_i$ uniformly at random from the interval $[1, R]$, where $R$ is some constant. We then pick $\Delta \pi_i$ randomly in the range $[\Delta t_i - \frac{R}{10}, \Delta t_i + \frac{R}{10}]$ where $\Delta \pi_i \geq 1$ for $i \in M$.

Also, for all the types mentioned above, we pick $\pi_d(i) = \gamma \Delta \pi_i$, where $\gamma$ is picked uniformly at random from the $[0.1, 0.4]$ interval. We also set $V_{min} = 18°\text{C}$, $V_{max} = 22°\text{C}$, and $A(0) = -3$. Based on this information, we can reverse engineer all the other parameters $A(i)$ and $\pi_c(i)$ for all $i \in M$ of this multi-mode system instance.

We tested our algorithms on randomly generated instances with strongly correlated, weakly correlated, and uncorrelated coefficients as defined in Section 5.5 of [32] for the 0-1 knapsack problem.

| | **Q=0.1** | | | **Q=1** | | | **Q=10** | | |
|---|---|---|---|---|---|---|---|---|---|
| | $T_{min}$ | $\overline{T}$ | $T_{max}$ | $T_{min}$ | $\overline{T}$ | $T_{max}$ | $T_{min}$ | $\overline{T}$ | $T_{max}$ |
| **n=2** | $10^{-6}$ | 0.007 | 0.03 | $10^{-6}$ | 0.008 | 0.03 | $10^{-6}$ | 0.015 | 0.05 |
| **n=4** | $10^{-6}$ | 0.013 | 0.032 | 0.015 | 0.035 | 0.078 | 0.015 | 0.038 | 0.094 |
| **n=6** | $10^{-6}$ | 0.021 | 0.079 | 0.031 | 0.063 | 0.125 | 0.031 | 0.613 | 1.141 |
| **n=8** | $10^{-6}$ | 0.03 | 0.11 | 0.047 | 0.091 | 0.141 | 0.047 | 0.105 | 1.479 |
| **n=10** | 0.015 | 0.043 | 0.125 | 0.078 | 0.122 | 0.266 | 0.062 | 0.147 | 2.316 |
| **n=20** | 0.062 | 0.163 | 0.343 | 0.172 | 0.830 | 7.028 | 0.203 | 1.556 | 20.993 |
| **n=30** | 0.313 | 0.467 | 0.672 | 0.406 | 2.188 | 11.677 | 0.328 | 2.400 | 28.602 |
| **n=40** | 0.688 | 0.946 | 1.453 | 0.516 | — | — | 0.453 | — | — |
| **n=50** | 0.961 | — | — | 0.782 | — | — | 0.578 | — | — |

TABLE 6.1: Average running time over 100 random strongly correlated instances for the optimal Integer Programming algorithm (in seconds), where $m_i$ is the number of modes.

For each instance we consider various lengths of the time horizon $t_{max} = h \cdot \sum_{i \in M} \Delta t_i$, where $h = \{0.1, 1, 10\}$. We tested our algorithm for different values of $R$, but since there was no significant difference in the relative performance of the algorithms, we only include the running times for $R = 10,000$.

Tables 6.1, 6.3 and 6.5 show the average execution time of the optimal integer linear programming (ILP) algorithm in milliseconds. As shown in Table 6.1, the dashed red cells mean that the algorithm suffered from a time out with 30 minutes execution period. It means that the integer programming algorithm could not find a solution within 30 minutes for at least one instance out of the 100 trials that were run every time. For these cells, we can not provide an average execution time or a maximum execution time either. Because of that I was not able to generate nice diagrams as in Section 6.2 and presented the results in tables form. Note that, for all the time measurements we included in this chapter, the function responsible for measuring the execution time of a program cannot measure time less than $10^{-6}$. If the program's execution time is less than $10^{-6}$, the function returns a value of 0. We replace every 0-value returned by the smallest execution time the function can measure which is $10^{-}6$.

| | **Q=0.1** | | | **Q=1** | | | **Q=10** | | |
|---|---|---|---|---|---|---|---|---|---|
| | $T_{min}$ | $\overline{T}$ | $T_{max}$ | $T_{min}$ | $\overline{T}$ | $T_{max}$ | $T_{min}$ | $\overline{T}$ | $T_{max}$ |
| **n=2** | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $3 \cdot 10^{-5}$ | 0.015 | $10^{-6}$ | 0.001 | 0.016 |
| **n=4** | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | 0.001 | 0.016 | $10^{-6}$ | 0.006 | 0.016 |
| **n=6** | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | 0.002 | 0.016 | $10^{-6}$ | 0.022 | 0.094 |
| **n=8** | $10^{-6}$ | $3 \cdot 10^{-4}$ | 0.015 | $10^{-6}$ | 0.005 | 0.016 | 0.031 | 0.058 | 0.156 |
| **n=10** | $10^{-6}$ | $5 \cdot 10^{-4}$ | 0.016 | $10^{-6}$ | 0.015 | 0.032 | 0.047 | 0.111 | 0.375 |
| **n=20** | $10^{-6}$ | 0.014 | 0.078 | 0.125 | 0.223 | 0.438 | 0.516 | 1.235 | 5.922 |
| **n=30** | 0.047 | 0.073 | 0.11 | 0.656 | 1.083 | 1.578 | 2.765 | 5.150 | 18.235 |
| **n=40** | 0.188 | 0.257 | 0.406 | 1.860 | 3.349 | 5.640 | 3.862 | 6.291 | 38.651 |
| **n=50** | 0.469 | 0.665 | 0.937 | 3.109 | 6.354 | 11.876 | 5.534 | 8.345 | 66.585 |

TABLE 6.2: Average running time over 100 random strongly correlated instances for the FPTAS approximation algorithm with 10% precision (in seconds), where $m_i$ is the number of modes.

Table 6.7 shows the average execution time of the two approximation algorithm in milliseconds for all the strongly correlated, weakly correlated, and uncorrelated data instances. The execution time is really small for all instances that we tried it on. Although this algorithm in general can return a solution with twice the optimal cost in the worst-case, by comparing its solutions with the optimal ones found by the IP algorithm, we found that, for all observed instances, the relative performance was below 10%. Moreover, as we showed in Section 3.5, the longer the time horizon is, the better are the worst-case guarantees that this algorithm provides. So if each heater has to be used at least 11 times by itself to cover the whole time horizon (i.e. $k_i \geq 11$ for all $i \in M$), the cost of the solution returned by this algorithm is at most $1/10 = 10\%$ higher than the optimal one.

Finally, Tables 6.2, 6.6, and 6.4 show the average execution time in milliseconds for the FPTAS approximation algorithm with $\rho = 10\%$ over the strongly correlated, weakly correlated, and uncorrelated knapsack data respectively. We found that the FPTAS quickly produces solutions to the instances, where the IP algorithm suffers from time-outs. The explanation behind this is the use of the scaling factors described in Section 4.5.2, which decreases the maximum cost considered

| | **Q=0.1** | | | **Q=1** | | | **Q=10** | | |
|---|---|---|---|---|---|---|---|---|---|
| | $T_{min}$ | $\overline{T}$ | $T_{max}$ | $T_{min}$ | $\overline{T}$ | $T_{max}$ | $T_{min}$ | $\overline{T}$ | $T_{max}$ |
| **n=2** | $10^{-6}$ | 0.007 | 0.031 | $10^{-6}$ | 0.009 | 0.032 | $10^{-6}$ | 0.014 | 0.094 |
| **n=4** | $10^{-6}$ | 0.013 | 0.063 | 0.015 | 0.026 | 0.063 | $10^{-6}$ | 0.030 | 0.048 |
| **n=6** | $10^{-6}$ | 0.022 | 0.047 | 0.015 | 0.041 | 0.094 | 0.016 | 0.056 | 0.125 |
| **n=8** | 0.015 | 0.030 | 0.063 | 0.031 | 0.058 | 0.109 | 0.031 | 0.074 | 0.208 |
| **n=10** | 0.015 | 0.04 | 0.079 | 0.046 | 0.074 | 0.109 | 0.047 | 0.086 | 0.144 |
| **n=20** | 0.063 | 0.18 | 0.484 | 0.125 | 0.244 | 0.453 | 0.125 | 0.246 | 0.691 |
| **n=30** | 0.219 | 0.413 | 0.687 | 0.203 | 0.431 | 0.765 | 0.203 | 0.437 | 0.734 |
| **n=40** | 0.258 | 0.626 | 1.002 | 0.344 | 0.653 | 1.234 | 0.343 | 0.709 | 1.304 |
| **n=50** | 0.428 | 0.874 | 1.494 | 0.422 | 0.931 | 2.109 | 0.438 | 0.989 | 2.656 |

TABLE 6.3: Average running time over 100 random uncorrelated instances for the optimal Integer Programming algorithm (in seconds), where $m_i$ is the number of modes.

and so also decreases the computation time.

## 6.4 Conclusions

Based on these tests we can conclude that, for multi-mode systems with small number of modes, the optimal integer programming algorithm as well as the FPTAS approximation algorithm run quickly and give the exact optimal schedule or one which is very close to optimal (in the case of using the FPTAS approximation algorithm), respectively. In all other instances, the constant factor approximation algorithm is the best choice, as it runs really quickly and most of the time gives a near-optimal solution.

| | **Q=0.1** | | | **Q=1** | | | **Q=10** | | |
|---|---|---|---|---|---|---|---|---|---|
| | $T_{min}$ | $\overline{T}$ | $T_{max}$ | $T_{min}$ | $\overline{T}$ | $T_{max}$ | $T_{min}$ | $\overline{T}$ | $T_{max}$ |
| **n=2** | $10^{-6}$ | $10^{-4}$ | 0.002 | $10^{-6}$ | $2 \cdot 10^{-4}$ | 0.003 | $10^{-6}$ | $3 \cdot 10^{-4}$ | 0.002 |
| **n=4** | $10^{-6}$ | $2 \cdot 10^{-4}$ | 0.001 | $10^{-6}$ | $8 \cdot 10^{-4}$ | 0.01 | $10^{-6}$ | 0.003 | 0.014 |
| **n=6** | $10^{-6}$ | $2 \cdot 10^{-4}$ | 0.001 | $10^{-6}$ | 0.002 | 0.012 | $10^{-6}$ | 0.009 | 0.034 |
| **n=8** | $10^{-6}$ | $2 \cdot 10^{-4}$ | 0.002 | $10^{-6}$ | 0.003 | 0.025 | 0.001 | 0.032 | 0.167 |
| **n=10** | $10^{-6}$ | $3 \cdot 10^{-4}$ | 0.002 | $10^{-6}$ | 0.002 | 0.017 | $10^{-6}$ | 0.064 | 0.273 |
| **n=20** | $10^{-6}$ | 0.002 | 0.016 | $10^{-6}$ | 0.042 | 0.203 | $10^{-6}$ | 0.514 | 2.485 |
| **n=30** | $10^{-6}$ | 0.005 | 0.11 | $10^{-6}$ | 0.195 | 1.078 | 0.016 | 2.357 | 17.624 |
| **n=40** | $10^{-6}$ | 0.005 | 0.062 | $10^{-6}$ | 0.621 | 3.984 | 0.062 | 5.485.22 | 44.052 |
| **n=50** | $10^{-6}$ | 0.011 | 0.187 | $10^{-6}$ | 1.355 | 4.954 | 0.343 | 15.094 | 84.658 |

TABLE 6.4: Average running time over 100 random uncorrelated instances for the FPTAS approximation algorithm with 10% precision (in seconds), where $m_i$ is the number of modes.

| | **Q=0.1** | | | **Q=1** | | | **Q=10** | | |
|---|---|---|---|---|---|---|---|---|---|
| | $T_{min}$ | $\overline{T}$ | $T_{max}$ | $T_{min}$ | $\overline{T}$ | $T_{max}$ | $T_{min}$ | $\overline{T}$ | $T_{max}$ |
| **n=2** | $10^{-6}$ | 0.008 | 0.046 | $10^{-6}$ | 0.01 | 0.047 | $10^{-6}$ | 0.017 | 0.047 |
| **n=4** | $10^{-6}$ | 0.013 | 0.046 | $10^{-6}$ | 0.031 | 0.067 | 0.015 | 0.03 | 0.07 |
| **n=6** | $10^{-6}$ | 0.027 | 0.085 | 0.007 | 0.048 | 0.109 | 0.014 | 0.045 | 0.094 |
| **n=8** | 0.015 | 0.041 | 0.121 | 0.016 | 0.061 | 0.133 | 0.016 | 0.058 | 0.140 |
| **n=10** | 0.015 | 0.059 | 0.148 | 0.024 | 0.066 | 0.151 | 0.011 | 0.067 | 0.147 |
| **n=20** | 0.078 | 0.154 | 0.281 | 0.094 | 0.169 | 0.312 | 0.063 | 0.160 | 0.297 |
| **n=30** | 0.188 | 0.286 | 0.453 | 0.172 | 0.274 | 0.5 | 0.157 | 0.255 | 0.422 |
| **n=40** | 0.263 | 0.396 | 0.765 | 0.234 | 0.383 | 0.703 | 0.234 | 0.374 | 0.735 |
| **n=50** | 0.359 | 0.526 | 1.041 | 0.344 | 0.514 | 1.047 | 0.344 | 0.473 | 0.828 |

TABLE 6.5: Average running time over 100 random weakly correlated instances for the optimal Integer Programming algorithm (in seconds), where $m_i$ is the number of modes.

| | **Q=0.1** | | | **Q=1** | | | **Q=10** | | |
|---|---|---|---|---|---|---|---|---|---|
| | $T_{min}$ | $\overline{T}$ | $T_{max}$ | $T_{min}$ | $\overline{T}$ | $T_{max}$ | $T_{min}$ | $\overline{T}$ | $T_{max}$ |
| **n=2** | $10^{-6}$ | $2 \cdot 10^{-4}$ | 0.016 | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $6 \cdot 10^{-4}$ | 0.016 |
| **n=4** | $10^{-6}$ | $2 \cdot 10^{-4}$ | 0.015 | $10^{-6}$ | $6 \cdot 10^{-4}$ | 0.016 | $10^{-6}$ | 0.004 | 0.016 |
| **n=6** | $10^{-6}$ | $5 \cdot 10^{-4}$ | 0.016 | $10^{-6}$ | 0.003 | 0.031 | $10^{-6}$ | 0.022 | 0.094 |
| **n=8** | $10^{-6}$ | $8 \cdot 10^{-4}$ | 0.016 | $10^{-6}$ | 0.012 | 0.093 | $10^{-6}$ | 0.061 | 0.562 |
| **n=10** | $10^{-6}$ | 0.001 | 0.016 | $10^{-6}$ | 0.022 | 0.063 | 0.015 | 0.135 | 0.875 |
| **n=20** | $10^{-6}$ | 0.013 | 0.265 | $10^{-6}$ | 0.373 | 1.968 | 0.094 | 1.849 | 41.694 |
| **n=30** | $10^{-6}$ | 0.067 | 0.25 | 0.063 | 1.823 | 7.203 | 0.891 | 5.863 | 69.32 |
| **n=40** | $10^{-6}$ | 0.267 | 1.125 | 0.063 | 5.560 | 20.735 | 1.953 | 15.299 | 99.735 |
| **n=50** | $10^{-6}$ | 0.619 | 8.860 | 0.172 | 9.961 | 365.05 | 5.641 | 27.322 | 125.447 |

TABLE 6.6: Average running time over 100 random weakly correlated instances for the FPTAS approximation algorithm with 10% precision (in seconds), where $m_i$ is the number of modes.

| | **Q=0.1** | | | **Q=1** | | | **Q=10** | | |
|---|---|---|---|---|---|---|---|---|---|
| | $T_{min}$ | $\overline{T}$ | $T_{max}$ | $T_{min}$ | $\overline{T}$ | $T_{max}$ | $T_{min}$ | $\overline{T}$ | $T_{max}$ |
| **n=2** | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ |
| **n=4** | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ |
| **n=6** | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ |
| **n=8** | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ |
| **n=10** | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ |
| **n=20** | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ |
| **n=30** | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ |
| **n=40** | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ |
| **n=50** | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ | $10^{-6}$ |

TABLE 6.7: Average running time (in seconds) over 100 random strongly correlated, uncorrelated, weakly correlated instances for the constant (Two) approximation algorithm, where $m_i$ is the number of modes.

# Chapter 7

# Conclusions and Future Work

## 7.1 Summary and Conclusions

Linear hybrid systems are computationally challenging. In particular, safety and reachability are undecidable already for three variables. We have identified the class of simple linear hybrid systems as a class that arises naturally when studying the optimal control of heating or cooling systems: there is only one continuous variable (the temperature in our setting) in addition to the time. Although it was to be expected that the optimal control for this model is decidable, the fact that this problem is both NP-complete and admits an FPTAS was not. Only a small number of NP-hard problems admit an FPTAS, i.e. can be approximated with relative precision $\rho$, in polynomial time in the size of the input and $1/\rho$. Most NP-hard problems can be shown to be inapproximable within a constant relative performance in polynomial time unless P=NP. The existence of FPTAS, besides offering a cheap approximation in every desired precision, often indicates that good standard solvers will normally behave well.

The example we considered as an application for the multi-mode system is the temperature control of one room (in the case of a single dimension system) or multiple rooms (in the case of a multi-dimension system) using Heating, Ventilation and Air-conditioning system (HVAC) while paying the lowest possible average cost. So, the aim is to find schedules to maintain the temperature in the comfort zone between $V_{min}$ and $V_{max}$ and select the schedule with the minimum average cost. We also interested in designing approximation algorithms that produce approximation solutions that are not far from the optimal one with low execution time.

We studied the optimisation problem of a simple subclass of multi-mode systems in only one dimension (one variable). We studied this simple subclass in

Chapter 3 with an idle mode (a cost free mode $m$ with $A(m) < 0$) and generalised this work in Chapter 4 where all kind of modes are permitted.

For the idle mode case, a motivating example is to maintain the temperature of one room within a comfort zone using heaters only. In order to increase the temperature, we turn on a heater and turn all the heaters off (*idle* mode) to decrease the temperature. We showed that the optimal control problem over an infinite time horizon has an easy (LogSpace) computational complexity. We studied the optimal control problem over a finite time horizon in detail by first proving the it is NP-hard problem using a reduction from the unbounded knapsack problem which is NP-hard (its optimisation version) or NP-complete (its decision version). Then we showed the optimal schedule pattern and proved that the optimal schedule consists of complete leaps and possibly the last incomplete one. We provided an optimal algorithm that uses the integer linear programming to find the optimal safe schedules. Also, we proposed two approximation algorithms that run in polynomial time. The first one is a constant factor approximation algorithm (two approximation) that keeps using only the mode with the smallest running cost per unit time to heat the room up from $V_{min}$ to $V_{max}$ followed by the idle mode that cools the room down from $V_{max}$ to $V_{min}$. This algorithm generates solutions with average cost that could reach twice the optimal cost. We proposed an FPTAS to be able to find arbitrary precise approximate solutions. The FPTAS reduces the problem to the 0-1 knapsack problem and uses a dynamic programming algorithm to solve it. The approximate solution generated by the FPTAS has average cost greater than or equal the optimal cost and less than or equal to $(1 + \epsilon)$ of the optimal cost where $\epsilon$ is the FPTAS precision. Solving the optimal control problem for multi-mode systems with relative performance $\rho$ takes $\mathcal{O}(\mathsf{poly}(1/\rho)\mathsf{poly}(\text{size of the instance}))$ time.

In Chapter 4 we studied the same problem in a more general setting. As a motivating example, the aim is to control the room temperature using heaters –to rise the temperature– and air-conditioners –to decrease the temperature. We showed that the optimisation problem still is NP-hard while the decision problem is NP-Complete. We also studied the pattern for optimal schedules. We showed that there always exists an optimal schedule that takes any form out of 44 different cases. Any schedule consists of three parts: head, leaps and tail sections. We presented cost-nonincreasing and safety-preserving operations that convert any schedule into one of 44 patterns. We showed that the head and tail parts together can not be longer than 5 timed actions. Like the system with the idle mode, we proved that finding an optimal safe infinite schedule for one-dimensional multi-mode systems can be computed in deterministic LogSpace. We presented

a constant factor approximation algorithm (three-approximation) which runs in $\mathcal{O}(|\mathcal{A}|^7)$ time. The algorithm tries all possible patterns for an optimal schedule and for the leaps section always picks leaps of the same type. It then adds, if necessary or for cost efficiency, a partial leap to the leaps section and minimises the total cost of the just constructed schedule by optimising the time duration of this partial leap. We showed that the cost minimisation problem for the general case one dimensional multi-mode systems has an FPTAS by a polynomial time reduction to the 0-1 Knapsack problem, for which many FPTAS algorithms exist. We then try all the 44 patterns and simply select the one with the minimal cost.

Going from the single dimension to the multi-dimension multi-mode system, we showed first that the optimal safe schedule may not exist. We overcome this problem by permitting a safety deviation $\epsilon$ from the comfort zone. We showed that finding an optimal limit-safe abstract schedule in $\mathcal{A}$ can be done in nondeterministic exponential time. We also showed that if a limit-safe abstract schedule exists, then finding an $\epsilon$-safe $\epsilon$-optimal strategy can be done in deterministic polynomial space and implemented an algorithm to find the $\epsilon$-safe $\epsilon$-optimal solution.

An implementation in Java was done for the the algorithms presented in Chapter 3 which studies the optimisation for the multi-mode single dimension system with an idle mode. We tested the performance of the integer programming, the two approximation algorithm and the FPTAS algorithms. We tested the algorithms performance over normal instances and hard knapsack instances. Based on these tests with the normal instances, we can conclude that, for multi-mode systems with a small number of modes, the optimal integer programming algorithm as well as the FPTAS approximation algorithm run quickly and give the exact optimal schedule or a very near optimal one (in the case of using the FPTAS approximation algorithm), respectively. In all other instances, the constant factor approximation algorithm is the best choice, as it runs really quickly and most of the time gives a near-optimal solution.

We have tested the three algorithms over hard knapsack instances: the strongly correlated, weakly correlated and uncorrelated instances. We found that the integer programming solver suffers from a time out with a time-out set to 30 minutes when we consider multi-mode systems with a large number of modes. So, we conclude that the integer programming solver is not suitable for solving the multi-mode systems with large number of modes and it is better to use the two-approximation algorithm which always produces an answer with a very low execution time while the FPTAS approximation produces better approximation in longer time.

## 7.2 Future Work

In the short term, we are going to design a tool using Java that implements the FPTAS approximation for the optimisation in the multi-mode multi-dimension system with discrete cost. The tool will be evaluated by considering a study case for controlling the temperature in multi-zone building. For the temperature control using HVAC system, we intend to study the problem more accurately by taking into account the parameters that affect the room temperature while modelling the systems such as the outside temperature and the heat transfer between rooms.

In the long term, we aim to modify our work to be suitable for minimising not only the average cost but also the peak demand in multi-dimensional systems. The cost of every mode will not be fixed any more. There will be categories for the cost being paid which depend on the peak power being consumed at every time t.

Finally, we want to address the problem of the day/night scheduling in which there exist two time sections (in the 24 hours). The system behaves differently in each time section. Each mode has different slopes and costs (the night electricity prices may be cheaper than the day prices) in each time section. Also, the comfort zone could be different. All these challenges make the problem harder and more challenging than what we solved in this thesis.

# Bibliography


[1] *EnergyPlus: Building energy simulation program*, `https://energyplus.net/`.

[2] *IBPT: International Building Physics Toolbox in Simulink*, `http://www.ibpt.org/`.

[3] *TRaNsient SYstems Simulation Program*, `http://sel.me.wisc.edu/trnsys/`.

[4] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, *The algorithmic analysis of hybrid systems*, Theoretical Computer Science **138** (1995), no. 1, 3–34.

[5] Rajeev Alur, Costas Courcoubetis, Thomas A Henzinger, and Pei-Hsin Ho, *Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems*, Hybrid systems, Springer, 1993, pp. 209–229.

[6] Rajeev Alur and David L. Dill, *A theory of timed automata*, Theoretical Computer Science **126** (1994), no. 2, 183–235.

[7] Rajeev Alur, Vojtech Forejt, Salar Moarref, and Ashutosh Trivedi, *Safe schedulability of bounded-rate multi-mode systems*, HSCC, ACM, 2013, pp. 243–252.

[8] Rajeev Alur, Vojtěch Forejt, Salar Moarref, and Ashutosh Trivedi, *Schedulability of bounded-rate multimode systems*, ACM Transactions on Embedded Computing Systems (TECS) **16** (2017), no. 3, 85.

[9] Rajeev Alur, Ashutosh Trivedi, and Dominik Wojtczak, *Optimal scheduling for constant-rate multi-mode systems*, Proceedings of the 15th ACM international conference on Hybrid Systems: Computation and Control, ACM, 2012, pp. 75–84.

[10] Sanjeev Arora, *Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems*, Journal of the ACM (JACM) **45** (1998), no. 5, 753–782.

[11] Eugene Asarin, Venkatesh P. Mysore, Amir Pnueli, and Gerardo Schneider, *Low dimensional hybrid systems – decidable, undecidable, don't know*, Information and Computation **211** (2012), 138–159.

[12] Richard Bellman, *Dynamic programming*, Courier Corporation, 2013.

[13] Devendra Bhave, Sagar Jha, Shankara Narayanan Krishna, Sven Schewe, and Ashutosh Trivedi, *Bounded-rate multi-mode systems based motion planning*, Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, ACM, 2015, pp. 41–50.

[14] Patricia Bouyer, *Weighted Timed Automata: Model-Checking and Games*, Electronic Notes in Theoretical Computer Science **158** (2006), 3–17.

[15] Patricia Bouyer, Thomas Brihaye, Marcin Jurdziński, Ranko Lazić, and Michał Rutkowski, *Average-Price and Reachability-Price Games on Hybrid Automata with Strong Resets*, Formal Modeling and Analysis of Timed Systems (Franck Cassez and Claude Jard, eds.), Lecture Notes in Computer Science, no. 5215, Springer Berlin Heidelberg, September 2008, pp. 63–77 (en).

[16] Mary Ann Branch and Andy Grace, *Matlab: Optimization toolbox: Computation, visualization, programming: User's guide, version 1.5*, MathWorks, 1996.

[17] Thomas Brihaye, Laurent Doyen, Gilles Geeraerts, Joël Ouaknine, Jean-François Raskin, and James Worrell, *On reachability for hybrid automata over bounded time*, International Colloquium on Automata, Languages, and Programming, Springer, 2011, pp. 416–427.

[18] Peter Bulychev, Alexandre David, Kim Gulstrand Larsen, Marius Mikučionis, Danny Bøgsted Poulsen, Axel Legay, and Zheng Wang, *UPPAAL-SMC: Statistical Model Checking for Priced Timed Automata*, Electronic Proceedings in Theoretical Computer Science, vol. 85, July 2012, pp. 1–16 (en).

[19] Andrew Chiu, George Davida, and Bruce Litow, *Division in logspace-uniform nc*, RAIRO-Theoretical Informatics and Applications **35** (2001), no. 3, 259–275.

[20] Stephen Cook, *The p versus np problem*, The millennium prize problems (2006), 87–104.

[21] Stephen A Cook, *The complexity of theorem-proving procedures*, Proceedings of the third annual ACM symposium on Theory of computing, ACM, 1971, pp. 151–158.

[22] George Dantzig, *Linear programming and extensions*, Princeton university press, 2016.

[23] Alexandre David, Dehui Du, Kim Guldstrand Larsen, Axel Legay, and Marius Mikučionis, *Optimizing Control Strategy Using Statistical Model Checking*, NASA Formal Methods (Guillaume Brat, Neha Rungta, and Arnaud Venet, eds.), Lecture Notes in Computer Science, no. 7871, Springer Berlin Heidelberg, May 2013, pp. 352–367 (en).

[24] Alexandre David, Peter Gjøl Jensen, Kim Guldstrand Larsen, Marius Mikučionis, and Jakob Haahr Taankvist, *Uppaal Stratego*, Tools and Algorithms for the Construction and Analysis of Systems (Christel Baier and Cesare Tinelli, eds.), Lecture Notes in Computer Science, no. 9035, Springer Berlin Heidelberg, April 2015, pp. 206–211 (en).

[25] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikučionis, and Zheng Wang, *Time for Statistical Model Checking of Real-Time Systems*, Computer Aided Verification (Ganesh Gopalakrishnan and Shaz Qadeer, eds.), Lecture Notes in Computer Science, no. 6806, Springer Berlin Heidelberg, July 2011, pp. 349–355 (en).

[26] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman, San Francisco, April 1979 (English).

[27] Oded Goldreich, *P, np, and np-completeness: The basics of computational complexity*, Cambridge University Press, 2010.

[28] I Gurobi Optimization, *Gurobi optimizer reference manual*, URL http://www.gurobi. com (2015).

[29] Thomas A Henzinger, *The theory of hybrid automata*, Verification of Digital and Hybrid Systems, Springer, 2000, pp. 265–292.

[30] John E Hopcroft, *Introduction to automata theory, languages, and computation*, Pearson Education India, 2008.

[31] Hans Kellerer, Ulrich Pferschy, and David Pisinger, *Introduction to np-completeness of knapsack problems*, Knapsack problems, Springer, 2004, pp. 483–493.

[32] _____, *Knapsack Problems*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004 (en).

[33] Jon Kleinberg and Eva Tardos, *Algorithm design*, Pearson Education India, 2006.

[34] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K Lenstra, Emmanuel Thomé, Joppe W Bos, Pierrick Gaudry, Alexander Kruppa, Peter L Montgomery, Dag Arne Osvik, et al., *Factorization of a 768-bit rsa modulus*, Annual Cryptology Conference, Springer, 2010, pp. 333–350.

[35] François Laroussinie, Nicolas Markey, and Ph Schnoebelen, *Model checking timed automata with one or two clocks*, CONCUR 2004-Concurrency Theory, Springer, 2004, pp. 387–401.

[36] Kim G Larsen, Marius Mikučionis, Marco Muniz, Jiří Srba, and Jakob Haahr Taankvist, *Online and compositional learning of controllers with application to floor heating*, International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2016, pp. 244–259.

[37] Bin Li and Andrew G. Alleyne, *Optimal on-off control of an air conditioning and refrigeration system*, American Control Conference (ACC), 2010, IEEE, 2010, pp. 5892–5897.

[38] CPLEX Users Manual, *Ibm ilog cplex optimization studio*, 1987.

[39] George B Mathews, *On the partition of numbers*, Proceedings of the London Mathematical Society **1** (1896), no. 1, 486–490.

[40] Kenneth L. McMillan and David K Probst, *A technique of state space search based on unfolding*, Formal methods in system design **6** (1995), no. 1, 45–65.

[41] Mahmoud AA Mousa, Sven Schewe, and Dominik Wojtczak, *Optimal control for simple linear hybrid systems*, 23rd International Symposium on Temporal Representation and Reasoning (TIME), IEEE, 2016, pp. 12–20.

[42] _____, *Optimal control for multi-mode systems with discrete costs*, International Conference on Formal Modeling and Analysis of Timed Systems, Springer, 2017, pp. 77–96.

[43] Truong X. Nghiem, Madhur Behl, Rahul Mangharam, and George J. Pappas, *Green scheduling of control systems for peak demand reduction*, Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on, IEEE, 2011, pp. 5131–5136.

[44] Truong X Nghiem, Madhur Behl, George J Pappas, and Rahul Mangharam, *Green scheduling for radiant systems in buildings*, Decision and Control (CDC), 2012 IEEE 51st Annual Conference on, IEEE, 2012, pp. 7577–7582.

[45] Truong X. Nghiem, George J. Pappas, and Rahul Mangharam, *Event-based green scheduling of radiant systems in buildings*, American Control Conference (ACC), 2013, IEEE, 2013, pp. 455–460.

[46] Frauke Oldewurtel, Andreas Ulbig, Alessandra Parisio, Göran Andersson, and Manfred Morari, *Reducing peak electricity demand in building climate control using real-time pricing and model predictive control*, Decision and Control (CDC), 2010 49th IEEE Conference on, IEEE, 2010, pp. 1927–1932.

[47] Christos H Papadimitriou, *The euclidean travelling salesman problem is np-complete*, Theoretical computer science **4** (1977), no. 3, 237–244.

[48] Luis Pérez-Lombard, José Ortiz, and Christine Pout, *A review on buildings energy consumption information*, Energy and buildings **40** (2008), no. 3, 394–398.

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

[50] Lieven MK Vandersypen, Matthias Steffen, Gregory Breyta, Costantino S Yannoni, Mark H Sherwood, and Isaac L Chuang, *Nmr quantum computing: Realizing shors algorithm*, Nature **414** (2001), 883–887.

[51] Vijay V Vazirani, *Approximation algorithms*, Springer Science & Business Media, 2013.

[52] Dominik Wojtczak, *Optimal Control for Linear-Rate Multi-mode Systems*, Formal Modeling and Analysis of Timed Systems (Víctor Braberman and Laurent Fribourg, eds.), Lecture Notes in Computer Science, no. 8053, Springer Berlin Heidelberg, August 2013, pp. 258–273 (en).

[53] ———, *On strong np-completeness of rational problems*, arXiv preprint arXiv:1802.09465 (2018).