University of Tennessee, Knoxville

TRACE: Tennessee Research and Creative Exchange

Masters Theses

Graduate School

8-2018

An Efficient Partial-Order Characterization of Admissible Actions for Real-Time Scheduling of Sporadic Tasks

Saajid Al Haque
University of Tennessee, shaque6@vols.utk.edu

Follow this and additional works at: https://trace.tennessee.edu/utk_gradthes

Recommended Citation

Haque, Saajid Al, "An Efficient Partial-Order Characterization of Admissible Actions for Real-Time Scheduling of Sporadic Tasks. " Master's Thesis, University of Tennessee, 2018.
https://trace.tennessee.edu/utk_gradthes/5119

This Thesis is brought to you for free and open access by the Graduate School at TRACE: Tennessee Research and Creative Exchange. It has been accepted for inclusion in Masters Theses by an authorized administrator of TRACE: Tennessee Research and Creative Exchange. For more information, please contact trace@utk.edu.

To the Graduate Council:

I am submitting herewith a thesis written by Saajid Al Haque entitled "An Efficient Partial-Order Characterization of Admissible Actions for Real-Time Scheduling of Sporadic Tasks." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Donatello Materassi, Major Professor

We have read this thesis and recommend its acceptance:

Seddik M. Djouadi, Husheng Li

Accepted for the Council:
Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

An Efficient Partial-Order Characterization of Admissible Actions for Real-Time Scheduling of Sporadic Tasks

A Thesis Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Saajid Al Haque

August 2018

© by Saajid Al Haque, 2018
All Rights Reserved.

Acknowledgments

I would like to thank my advisor Dr. Materassi for his help and guidance while working on this thesis. I would also like to express my gratitude to him for providing me this opportunity to be a graduate research assistant. I would like to thank and acknowledge ARPA-E for supporting this work through the project titled *Robust Distributed Framework for Flexible Power Grids*. Furthermore, I would like to thank Dr. Djouadi and Dr. Li for taking the time to be on my committee and their insights for my thesis. In addition, I want to thank all the professors and staff that have helped me learn and grow during my time at the University of Tennessee, Knoxville.

A few sections of this thesis are similar to the material in the conference paper [1], and I would like to thank the other authors of the conference paper [1].

Abstract

In many scheduling problems involving tasks with multiple deadlines, there is typically a large degree of flexibility in determining which tasks to serve at each time step. Given a cost function it is often possible to cast a scheduling problem as an optimization problem to obtain the most suitable schedule. However, in several applications, especially when the schedule has to be computed in-line or periodically adjusted, the cost function may not be completely known a priori but only partially. For example, in some applications only the cost of the current allocation of resources to the tasks could be available. Under this scenario, a sensible approach is to optimally allocate resources without compromising the schedulability of the tasks. This work follows this approach by introducing a notion of partial ordering on the set of feasible schedules. This partial ordering is particularly useful to characterize which allocations of resources at the current time preserve the feasibility of the problem in the future. This enables the realization of fast algorithms for real-time scheduling. The model and algorithm presented can be utilized in different applications such as electric vehicle charging, cloud computing and advertising on websites. [1]

Table of Contents

1 Introduction **1**

1.1 Existing Scheduling Algorithms . . . 2

1.2 Potential Applications . . . 2

1.2.1 Electric Vehicle Charging . . . 2

1.2.2 Cloud Computing . . . 3

1.2.3 Advertising on Websites . . . 3

2 Problem Formulation **5**

3 Partial Ordering with Slack Vectors **10**

3.1 Notion of Slack . . . 10

3.2 Test Algorithm . . . 12

3.3 Slack Theorem . . . 13

4 Unique Maximal Slack Element **17**

4.1 Unique Maximal Slack Element Theorem . . . 17

4.2 Examples . . . 19

4.2.1 Infinity Example . . . 19

4.2.2 Scalar vs. Vector Example . . . 19

4.2.3 Auction Example . . . 22

4.3 Incoming Arrivals . . . 24

5 Proving The Unique Maximal Slack Element Theorem **25**

5.1 Common and Difference Tasks . . . 25

5.2 Minimum and Maximum Action 26
5.3 Two Actions That Differ in One Task 27
5.4 Corollary to Two Actions Differing in One Task 28
5.5 Actions That Are Not Slack-Comparable 29
5.6 Actions That Are Not Slack-Comparable and Differ in Two Tasks 30
5.7 Maximal Element When Two Actions Serve The Same Number Of Tasks . . 32
5.8 Proof for Theorem 16 34

6 Conclusion **37**

Bibliography **38**

Appendix **42**

A Proving Theorem 23 43

Vita **54**

List of Figures

3.1 Switch Part A 14

3.2 Switch Part B 14

4.1 Infinity Example Schedule 20

4.2 Scalar vs. Vector Example 21

4.3 Auction Example 23

5.1 Lemma 20 Visualization. 28

5.2 Lemma 22 Visualization. 31

5.3 Phantom Task Schedule 36

A.1 Case 10 Visualization. 48

A.2 Case 11 Visualization. 49

A.3 Case 16 Visualization. 50

A.4 Case 17 Visualization. 51

A.5 Case 22 Visualization. 53

Chapter 1

Introduction

Scheduling is a decision-making process that allocates resources to tasks over a period of time and in most cases, attempts to optimize their completion according to a given objective function [2]. The resources and tasks can be of different nature based on the application, but the mathematical models used to describe a scheduling problem share many characteristics. The general problem consists of a set of tasks with potentially heterogeneous deadlines, where all of the tasks cannot be attended to at the same time or instantaneously. Consequently, a schedule is essential to ensure that all of the tasks are completed before their deadlines. Linear programming is a prevalent tool to compute schedules and can also be used to optimize with respect to a linear cost function [2]. Other optimization techniques are, of course, also possible [3], [4]. In some applications, while the constraints are predetermined (such as the demand and deadline), the cost function may not be completely known a priori, either because it depends on exogenous factors, or because of privacy concerns in multi-agent scenarios. Indeed the overall cost function might be given by the sum of the costs of the specific actions taken at each time t, while the cost of future actions is not known. Since the space of current actions is smaller than the space of all schedules, a characterization of all admissible actions can lead to a computationally efficient algorithm. The algorithm presented optimizes with respect to the current action instead of the global schedule while guaranteeing that the deadlines will still be met. [1]

1.1 Existing Scheduling Algorithms

Scheduling methods can be typically grouped into static and dynamic priority scheduling algorithms, and this thesis is related to the area of dynamic priority scheduling. One common dynamic algorithm is the least slack time (LST), and for a certain subclass of dynamic scheduling problems such as a single processor system, the LST is a possible optimal algorithm [5]. This algorithm is a deadline based priority strategy that gives the highest priority to the tasks with the least slack (maximum amount of time that can pass until the task must be served to meet its deadline). One natural question that arises after studying this method is if all the different combinations of tasks that can be served while keeping feasibility can be described using the notion of slack. This article will introduce a partial order that compares sets of tasks and uses the partial ordering to investigate whether a unique maximal slack exists that retains feasibility.

1.2 Potential Applications

Potential applications span several domains.

1.2.1 Electric Vehicle Charging

In the power grid, a fundamental concept is that supply must equal the demand constantly. Traditionally, the output of the generators is assumed to be controllable while user demand is treated to be inelastic [6]. However, the addition of wind and solar to the grid challenges the convention that generation must satisfy all loads instantaneously. The variability of alternative energy sources poses an obstacle to transitioning to a grid more reliant on renewable energy. A strategy being explored changes the idea that the loads are rigid and treats them as deferrable. For example, electric vehicle batteries and many other loads can be considered to be flexible.

Various studies have been completed recently on supplying flexible loads with different demand side schemes. Heuristic scheduling algorithms such as least slack first and earliest deadline first are presented in [7] to demonstrate the possible scheduling of deferrable loads to

utilize renewable energy sources. An energy delivery management method is developed in [8] that shows consumers may receive a significant discounted price for EV charging by delaying consumption. Other approaches [9] assume partial information given by the consumer and create an efficient pricing method that incentives customers to reveal their true deadlines regardless of actions by others. Furthermore, a non-causal algorithm is presented in [10] that fulfills all the energy requirements for a group of deferrable loads without surplus or deficit. The model presented in this article can be generally applied to a group of flexible loads that are connected to the same bus so that there is an aggregate limit on how many tasks can be served at the same time. [1]

1.2.2 Cloud Computing

Another problem that may fall under the model of this article is real-time scheduling of tasks in the cloud [11]. The problem is to execute a large number of time-sensitive tasks that need to be completed with limited computational power, and a schedule can lead to efficient use of the resources by determining the use of the processors and the priority of the tasks [12]. A general method to derive schedulability conditions for these types of systems is presented in [13] where real-time tasks are considered with deadlines on a platform of identical processors. A task splitting approach is explored in [14] for scheduling sporadic task systems on multiprocessor platforms. The authors of [11] utilize an algorithm combining priority based and EDF techniques to provide better performance by reducing the total execution time of the tasks in cloud computing. The method presented in this thesis is different in that all of the possible choices at the current time t can be determined, and then a suitable action can be chosen to move forward with. [1]

1.2.3 Advertising on Websites

Recently web advertising has grown significantly into one of the largest forms of advertising. While this source of income continues to grow, advertisers are placing more emphasis on making their advertisements more efficient and targeted. Websites want to reach specific groups with products that they are more likely to be interested in. This type of product

placement not only benefits the advertisers, but also enhances the users experience while on that website. The objective function for scheduling web advertising could be based on identification characteristics such as user keyword searches. One way advertisements are sold to publishers is through the traditional guaranteed market. In this scheme, advertisers make a guaranteed contract with a website that commits to delivering a predetermined number of advertisements within a certain period of time [15]. Furthermore, [15] addresses the scheduling and capacity problem for certain cases of dealing with multiple guaranteed contracts with different deadlines. Other work [16] attempts to ensure a unique number of consumers see the advertisements while still ensuring the advertisements are shown the predetermined number of times. A scheduling model could potentially be applied to this type of advertising scheme to increase the success of the advertisements. [17]

Chapter 2

Problem Formulation

A problem setup is considered that is analogous to the one described in [1] and [18]. First, a definition for a task is introduced.

Definition 1 (Task)**.** A task B is a 3-tuple $(D, T, \overline{u}) \in \mathbb{R}^3$ where $D > 0$ defines the demand, T is the deadline, i.e. the time left to satisfy such demand, and $\overline{u} > 0$ is the maximum serving rate.

In line with many practical applications, and with minimal loss of generality, a discretization of time and of the parameters is introduced that defines the tasks. A unitary sampling time is assumed and the following definition is introduced.

Definition 2 (Integer task)**.** A task $(D, T, \overline{u})$ is *integer* if $T \in \mathbb{N}$ and if $\frac{D}{\overline{u}} \in \mathbb{N}$.

The following discrete time model is introduced to describe the dynamic process of task completion

$$x_{k+1} = x_k + u_k$$
$$x_0 = -D$$

where $x_k \leq 0$ is the state of completion of the task, and $u_k \geq 0$ is the rate at which the task is served (in a piece-wise constant way) during the k-th interval, namely between time k and $k + 1$.

The constraint on the maximum rate and the deadline for task completion correspond respectively to the inequalities

$$0 \leq u_k \leq \overline{u}, \quad \text{for } k = 0, \ldots, T-1,$$

and the equality

$$x_T = 0.$$

However, for practical purposes the case of multiple tasks is considered, the behaviors of which are coupled by the fact that, at any time, the aggregate serving rate (the sum of the rates at which different tasks are served) cannot exceed a given limit $\overline{U}$.

Therefore, consider N tasks $(D^{(i)}, T^{(i)}, \overline{u}^{(i)})$, where $i = 1, ..., N$, and the following problem is examined.

Problem 3 (Task scheduling problem)**.** The task scheduling problem is defined as the problem of deciding the variables $u_k^{(i)}$ for $k = 0, ..., T := \max_{1 \leq i \leq N} T^{(i)}$ such that for any task $i = 1, ..., N$

$$\begin{aligned} x_{k+1}^{(i)} &= x_k^{(i)} + u_k^{(i)} \\ x_0^{(i)} &= -D^{(i)} \\ x_{T^{(i)}}^{(i)} &= 0 \end{aligned} \tag{2.1}$$

where $x_k^{(i)} \leq 0$ is the state of task i at time k, and such that, at any time k,

$$\begin{aligned} &0 \leq u_k^{(i)} \leq \overline{u}^{(i)} \quad \text{for } i = 1, ..., N \\ &\sum_{i=1}^{N} u_k^{(i)} \leq \overline{U}. \end{aligned} \tag{2.2}$$

The horizon of the problem is refereed to as $[0, T]$.

The following definitions are introduced as well.

Definition 4 (Action)**.** The *action* at time k is defined as the vector in $u_k = (u_k^{(1)} ... u_k^{(N)})$ obtained by stacking the rates of the different tasks.

Definition 5 (Feasible schedule)**.** Given the tasks $(D^{(i)}, T^{(i)}, \overline{u}^{(i)})$ for $i = 1, ..., N$, and an aggregate limit $\overline{U}$, a schedule (i.e. a sequence of actions) $\{u_k\}_{k=0}^{T-1}$ is *feasible* if it solves the task scheduling problem, i.e. it satisfies the constraints (2.1) and (2.2).

Definition 6 (Schedulable tasks)**.** Given an aggregate limit $\overline{U}$, the tasks $(D^{(i)}, T^{(i)}, \overline{u}^{(i)})$ for $i = 1, ..., N$, are *schedulable* if a feasible schedule for the corresponding task scheduling problem exists.

The goal of the analysis presented in this thesis is to give necessary and sufficient conditions so that the actions of the users meet the constraints of the task scheduling problem. While implementing a schedule in real time, it is of paramount importance to choose an appropriate action that does not compromise feasibility of the schedule in the future.

The analysis is performed at time $k = 0$ and provides guarantees that, if the tasks are schedulable to begin with, then a certain action u_0 leaves the tasks in a schedulable configuration over the horizon $[1, T]$. Then the same approach can be iteratively applied.

An effective characterization of the set of all (and only) the admissible actions allows one to determine an optimal action according to some cost criteria.

Consequently, the following definitions are introduced.

Definition 7 (Admissible action)**.** Given the tasks $(D^{(i)}, T^{(i)}, \overline{u}^{(i)})$ for $i = 1, ..., N$, and an aggregate limit $\overline{U}$, an action

$$u_0 = \begin{bmatrix} u_0^{(1)} \\ \vdots \\ u_0^{(N)} \end{bmatrix}$$

is *admissible* if:

- it satisfies

$$0 \leq u_0^{(i)} \leq \overline{u}^{(i)},\ i = 1, \ldots, N, \quad \text{and} \quad \sum_{i=1}^{N} u_0^{(i)} \leq \overline{U},$$

- the tasks $(D^{(i)} - u_0^{(i)}, T^{(i)} - 1, \overline{u}^{(i)})$ are schedulable with the global constraint $\overline{U}$.

A technical assumption is made with minimal loss of generality.

Assumption 8. The tasks $(D^{(i)}, T^{(i)}, \overline{u}^{(i)})$, $i = 1, ..., N$, are all integer tasks. Moreover, there exist a common $\overline{u}$, that we denote as the *unit rate*, such that

$$\frac{D^{(i)}}{\overline{u}} \in \mathbb{N}, \quad \frac{\overline{u}^{(i)}}{\overline{u}} \in \mathbb{N}, \quad \frac{\overline{U}}{\overline{u}} \in \mathbb{N},$$

for all $i = 1, ..., N$.

Based on this, another definition is introduced.

Definition 9 (Integer schedule)**.** Given a task scheduling problem, a schedule is integer if all its elements are integer multiples of a unit rate $\overline{u}$.

In this thesis, *integer schedules* will exclusively be focused on. This choice is reasonable in the common practice, where most of the time tasks can be served at some quantized rate of service (i.e. according to the number of processors assigned to the thread in a multiprocessor system [2], or according to some of the EV battery charging standards where different levels of charging speed are available).

It is worth remarking that, even if the problem of finding a solution to the scheduling problem (2.1)-(2.2) becomes easier if the integer constraints are relaxed (and reduces to the application of linear programming to load scheduling [19, 20]), a practical characterization of all the feasible actions remains a difficult problem. On the other hand, the following result shows that the restriction to integer schedules has no effect on the schedulability analysis of the task scheduling problem.

Theorem 10. Consider a set of N tasks $(D^{(i)}, T^{(i)}, \overline{u}^{(i)})$, with an aggregate bound $\overline{U}$, and let Assumption 8 hold. The tasks are schedulable if and only if an integer schedule exists.

Proof. To prove Theorem 10, we will use the theorem that states if A is a totally unimodular matrix while vectors b, c and constant u are integer, then the linear program

$$\max cq \quad \text{subject to} \quad Aq = b \quad \text{and} \quad 0 \leq q \leq u \tag{2.3}$$

will have an integer solution for q under the assumption that a solution exists [21]. The scheduling problem described in the problem formulation can be described with the

following equations

$$\sum_{k=0}^{T^{(i)}-1} u_k^{(i)} = D^{(i)} \quad \text{for} \quad i = 1, \ldots, N$$

$$y_k^{(1)} + \ldots + y_k^{(\overline{U})} + \sum_{i=1}^{N} u_k^{(i)} = \overline{U} \quad \text{for} \quad k = 0, \ldots, T-1.$$

where the variables y are auxiliary variables to ensure that the equality for the consumption is satisfied. These equations can be written in the form of a linear program (Equation 2.3) where the constant u is 1 and A and b are selected according to

$$q = \left[u_0^{(1)} ... u_0^{(N)} u_1^{(1)} ... u_{T-1}^{(N)} y_0^{(1)} ... y_{T-1}^{(\overline{U})} \right]'.$$

From this description, it should be clear that b will always be integer because of Assumption 8. Now we need to show that the matrix A will be totally unimodular. The A matrix will consist of 0 and 1 entries and have at most two non-zero entries in each column. Consequently, A will be totally unimodular according to a Theorem by Heller and Tompkins [21]. The conditions have been satisfied for the linear program formulation of the scheduling problem to have integer solutions. Therefore, if the tasks are schedulable, a feasible integer schedule will also exist. □

Moreover, there is no loss of generality by assuming that all tasks have the same rate constraint $\overline{u}^{(i)} = 1$, since the general case can always be reformulated in this way, by "splitting" each task into smaller tasks with homogeneous rate constraints [18].

Definition 11 (Aggregate serving rate)**.** Given an action u_0, the *aggregate serving rate* is indicated as

$$\|u_0\|_1 = \sum_{i=1}^{N} u_0^{(i)}. \tag{2.4}$$

Chapter 3

Partial Ordering with Slack Vectors

3.1 Notion of Slack

The notion of slack from the area of multiprocessor scheduling [2] is used to create a partial ordering relation.

Definition 12 (Slack)**.** The *slack* of a task at time 0 is defined as

$$slack(B^{(i)}) = s^{(i)} := T^{(i)} - \frac{D^{(i)}}{\overline{u}^{(i)}}. \tag{3.1}$$

If the serving rate is assumed to be unitary $\overline{u}^{(i)} = 1$, Eq. 3.1 reduces to

$$slack(B^{(i)}) = s^{(i)} := T^{(i)} - D^{(i)}. \tag{3.2}$$

The slack represents the maximum idle time that the task can wait, before having to be necessarily served, in order to meet its deadline.

Given a schedule $\{u_l\}_{l=0}^{k-1}$, we can find the slack of a task at time k (assuming $\overline{u}^{(i)} = 1$) with

$$slack_u(B^{(i)}, k) := (T^{(i)} - k) - (D^{(i)} - \sum_{l=0}^{k-1} u_l^{(i)}). \tag{3.3}$$

Based on this definition, a function $SlackVector$ is introduced on the action u_k.

Definition 13 (Slack Vector)**.** *$SlackVector(\cdot)$ maps $u \in \{0,1\}^N$ into a vector of $\overline{U}$ entries that can either be integers or ∞*

$$SlackVector : \{0,1\}^N \rightarrow (\mathbb{N} \cup \{\infty\})^{\overline{U}}. \tag{3.4}$$

Let j be the number of tasks that are served in u_k. The first j entries of $SlackVector(u_k)$ are the ordered slacks of the j tasks being served and the remaining $(\mid \overline{U} \mid -j)$ entries are ∞.

As an example, consider the case of an action serving tasks 2 and 3 among 5 tasks and $\overline{U} = 3$,

$$SlackVector\left(\begin{bmatrix} 0\\1\\1\\0\\0 \end{bmatrix}\right) = \begin{bmatrix} s^{(2)} \\ s^{(3)} \\ \infty \end{bmatrix}$$

under the assumption that $s^{(2)} \leq s^{(3)}$.

It is possible to define a *partial ordering* relation between vectors of dimension $\overline{U}$ via the following definition.

Definition 14 (Slack partial ordering)**.** Let $\mathbf{s}'$ and $\mathbf{s}''$ be two slack vectors with dimension d. We say that

$$\mathbf{s}' \preccurlyeq \mathbf{s}''$$

if the ith entry of $\mathbf{s}'$ is less than or equal to the ith entry of $\mathbf{s}''$ for $i = 1, ..., d$.
Consequently, two vectors are not slack-comparable if neither of the following hold

$$\mathbf{s}' \preccurlyeq \mathbf{s}''$$

$$\mathbf{s}' \succcurlyeq \mathbf{s}''.$$

As an example, $\left[\begin{smallmatrix} 1\\3 \end{smallmatrix}\right] \preccurlyeq \left[\begin{smallmatrix} 2\\4 \end{smallmatrix}\right]$, while $\left[\begin{smallmatrix} 1\\4 \end{smallmatrix}\right]$ and $\left[\begin{smallmatrix} 2\\3 \end{smallmatrix}\right]$ are not slack-comparable.

3.2 Test Algorithm

In addition, an algorithm is taken from [18] that tests the schedulability of a given sets of tasks. This algorithm will return true if a given set of tasks are schedulable.

Minimum Effort Algorithm

INPUT

- tasks: $(D^{(i)}, T^{(i)}, 1)$, $i = 1, ..., N$
- aggregate limit: $\overline{U}$

EXECUTE

1. INITIALIZE
    - $T := \max_{1 \leq i \leq N} T^{(i)}$
    - $u_k^{(i)} = 0$ for $0 \leq k < T$, $1 \leq i \leq N$
2. FOR $k = T - 1 : -1 : 0$
    - Create **ActiveTasks**:= list of all tasks with deadline $T^{(i)} > k$ and demand $D^{(i)} > 0$
    - Compute $N_A :=$ # of elements in **ActiveTasks**
    - Order **ActiveTasks** according to their *reverse* slack $r^{(i)} := k + 1 - D^{(i)}$
    - For the first $\min(N_A, \overline{U})$ tasks with least reverse slack $r^{(i)}$ in **ActiveTasks**, decrease $D^{(i)}$ by 1 and assign $u_k^{(i)} := 1$

OUTPUT:

- schedulability: if $D^{(i)} = 0$ for all $i = 1, ..., N$ return **TRUE** otherwise return **FALSE**
- schedule: $u_k^{(i)} = 0$ for $0 \leq k < T$, $1 \leq i \leq N$
- effort: $\underline{U} := \sum_{i=1}^{N} u_0^{(i)}$

3.3 Slack Theorem

Given an admissible action, the following result (taken from [18]) shows that it is possible to use the slack partial ordering to find other admissible actions.

Theorem 15. Consider the tasks $B^{(i)} = (D^{(i)}, T^{(i)}, 1)$ for $i = 1, ..., N$, together with the aggregate constraint $\overline{U}$, and let Assumption 8 hold. Let u_0 be an admissible action. Then any action v_0 is also admissible if

$$SlackVector(v_0) \preccurlyeq SlackVector(u_0).$$

In other words, Theorem 15 states that if an action is admissible, then an action smaller in slack, in the sense provided by the relation "$\preccurlyeq$", is also admissible.

Proof. Assume that there is a feasible integer scheduling where $u_0^{(i_1)} = \ldots = u_0^{(i_M)} = 1$, Let j be a task such that $u_0^{(j)} = 0$ and $s^{(j)} \leq s^{(i_m)}$ for some $1 \leq m \leq M$. We will first prove that the decision where

$$\begin{aligned} v_0^{(i_1)} &= \ldots = v_0^{(i_{m-1})} = v_0^{(i_{m+1})} = \ldots = v_0^{(i_M)} = 1 \\ v_0^{(i_m)} &= 0 \\ v_0^{(j)} &= 1 \\ v_0^{(i)} &= 0 \qquad \text{for } i \neq j, i_1, ..., i_M. \end{aligned}$$

is admissible.

Without any loss of generality assume $m = M$. Let us consider first the case where $T^{(i_M)} \leq T^{(j)}$. If $T^{(i_M)} \leq T^{(j)}$, and since $s_0^{(j)} \leq s_0^{(i_M)}$, there is a time k such that $0 < k \leq T^{(i_M)}$ and $u_k^{(i_M)} = 0$, $u_k^{(j)} = 1$. Thus, it is possible to make the j-th task to be served in place of the i_M-th one at time 0 and vice-versa at time k without violating the constraints of the problem, as depicted in the following figure (3.1).

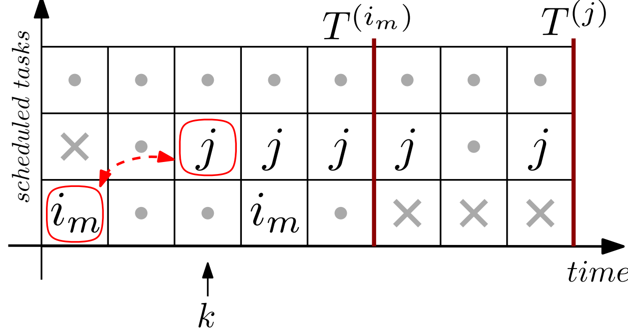

Figure 3.1: Switch Part A

Now, let $T^{(i_M)} > T^{(j)}$. If there is a time $0 < k \leq T^{(i_M)}$ where $u_k^{(i_M)} = 0$ and $u_k^{(j)} = 1$, we can operate the same switch as before. Otherwise, since $u_0^{(j)} = 0$, there must be a time $0 < k' \leq T^{(j)}$ where both the j-th and the i_M-th batteries consume, namely $u_{k'}^{(i_M)} = 1$, $u_{k'}^{(j)} = 1$. Since $T^{(i_M)} > T^{(j)}$, there also must be a time $T^{(j)} < k'' \leq T^{(i_M)}$ where neither the j-th nor the i_M-th batteries consume. Now move the consumption of the i_M-th battery from k' to k''. This is always possible: if at time k'' we have that $\sum_{i=1}^{N} u_{k''} < \overline{U}$ no constraint is violated, otherwise it is possible to find a battery $j' \neq j, i_M$ such that $u_{k''}^{(j')} = 1$, $u_{k'}^{(j')} = 0$ and anticipate its consumption at time k'. After this switch we have that at time k' the i_M-th battery does not consume while j-th does. Switching consumption beetween the two batteries at time 0 and k' is now possible without violating any constraint and the assertion is proved. This scenario is depicted in the following figure (3.2).

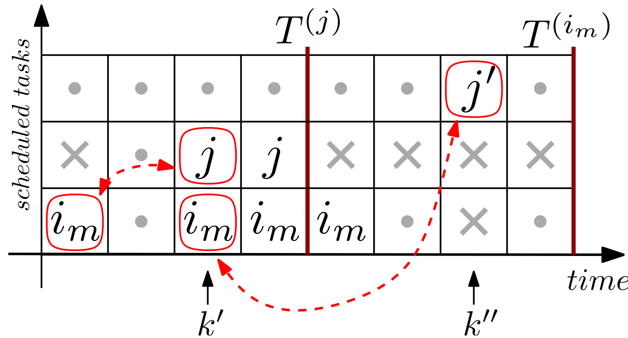

Figure 3.2: Switch Part B

Iterated application of the same reasoning allow to prove that any decision $\mathbf{v}_0$ such that $\|\mathbf{v}_0\|_1 = \|\mathbf{u}_0\|_1$ and such that $SlackVector(\mathbf{v}_0) \preccurlyeq SlackVector(\mathbf{u}_0)$, is also admissible (as it is the first decision in a feasible schedule). Finally, we conclude that this is also true in

the case in which $\|\mathbf{v}_0\|_1 > \|\mathbf{u}_0\|_1$ (and therefore in all cases in which $SlackVector(\mathbf{v}_0) \preccurlyeq SlackVector(\mathbf{u}_0)$), because that simply means that other tasks have been served at time 0, together with the ones served by the feasible schedule, necessarily yielding another feasible schedule. □

One direct consequence of Theorem 15 is that the Least Slack Time (LST) Algorithm can be used as another feasibility test for the specific scheduling problem described in the problem formulation. Indeed if there is an admissible action, the least slack action will be an admissible action as well because it will be "$\preccurlyeq$" than all other admissible actions. If a schedule with the initial action that has the least slack vector is not admissible, then no initial action will be admissible and the group of tasks will not be schedulable.

Least Slack Time (LST) Algorithm

INPUT

- tasks: $(D^{(i)}, T^{(i)}, 1)$, $i = 1, ..., N$
- aggregate limit: $\overline{U}$

EXECUTE

1. INITIALIZE
   - $T := \max_{1 \leq i \leq N} T^{(i)}$
   - $u_k^{(i)} = 0$ for $0 \leq k < T$, $1 \leq i \leq N$
2. FOR $k = 0 : 1 : T - 1$
   - Create **ActiveTasks**:= list of all tasks with deadline $T^{(i)} > k$ and demand $D^{(i)} > 0$
   - Compute $N_A :=$ # of elements in **ActiveTasks**
   - Order **ActiveTasks** according to their slack $s^{(i)} := T^{(i)} - D^{(i)} - k$
   - For the first $\min(N_A, \overline{U})$ tasks with least slack $s^{(i)}$ in **ActiveTasks**, decrease $D^{(i)}$ by 1 and assign $u_k^{(i)} := 1$

   OUTPUT:

   - schedulability: if $D^{(i)} = 0$ for all $i = 1, ..., N$ return **TRUE** otherwise return **FALSE**
   - schedule: $u_k^{(i)}$ for $0 \leq k < T$, $1 \leq i \leq N$

Chapter 4

Unique Maximal Slack Element

4.1 Unique Maximal Slack Element Theorem

Using the partial ordering relation described in the previous chapter, a maximal slack vector can be found to help determine all the admissible actions [1].

Theorem 16. Given a feasible scheduling problem $\mathcal{P}$, all the maximal actions have the same slack vector.

Proof. See Chapter 5. □

From Theorem 16, it is known that all the maximal admissible actions have the same slack vector $\hat{\mathbf{s}}$. From the knowledge of this unique slack vector it becomes possible to test if any given action u_0 is admissible. Indeed, $SlackVector(u_0) \preccurlyeq \hat{\mathbf{s}}$ if and only if u_0 is admissible. Thus, the knowledge of $\hat{\mathbf{s}}$ completely characterizes the set of all possible admissible actions. The problem then becomes to find $\hat{\mathbf{s}}$ in an efficient way.

Maximal Element Algorithm

INPUT

- The ordered set of all tasks in ascending order of slack augmented with $\overline{U}$ auxiliary tasks $\{B^{(1)}, B^{(2)}, ..., B^{(N)}, B^{(N+1)}, ..., B^{(N+\overline{U})}\}$.
- $B^{(N+1)}, ..., B^{(N+\overline{U})}$ are special tasks such that $slack(B^{(N+1)}) = ... = slack(B^{(N+\overline{U})}) = \infty$.

EXECUTE

- INITIALIZE
  - Let action $u = \{B^{(i_1)}, B^{(i_2)}, ..., B^{(i_{\overline{U}})}\}$ and $i_1 = 1, i_2 = 2, ..., i_{\overline{U}} = \overline{U}$.
  - Let $i_{\overline{U}+1} = N + \overline{U} + 1$
- FOR $d = \overline{U} : -1 : 1$
  - Let j = d
  - WHILE (u is admissible AND $\|u\| \geq d$ AND $j + 1 < i_{d+1}$)
    * Let $\{B^{(i_1)}, B^{(i_2)}, ..., B^{(i_{\overline{U}})}\}$ be the tasks served in u in ascending order of slack.
    * Take action u and set task $B^{(i_d)}$ to $B^{(j+1)}$.
    * IF ($slack(B^{(j+1)}) == \infty$) THEN remove $B^{(i_d)}$ in action u.
    * j = j + 1
  - Check the admissibility of action u.
  - IF (u is not admissible) THEN take action u and set task $B^{(i_d)}$ to $B^{(j-1)}$.

  OUTPUT:

- $\hat{\mathbf{s}}$ = the slack vector of u.

4.2 Examples

4.2.1 Infinity Example

A schedule in Figure 4.1 is shown with the initial action serving $B^{(3)}$ and $B^{(4)}$ corresponding to a slack vector of $[1, 1, \infty]^\intercal$. A gray box indicates that at time 0 the task i is served ($u_k^{(i)} = 1$) while a white box indicates that the task is idle ($u_k^{(i)} = 0$). The shaded boxes show a possible projected schedule for the future providing that the chosen action at time 0 is admissible. Each box also contains the numerical values of $D^{(i)}$. As all the demands are served for all tasks after the iteration at time $k = 4$, the problem is schedulable.

The problem in Figure 4.1 shows an example where the maximal slack vector for the initial action does not reach the aggregate bound. The aggregate bound is $\overline{U} = 3$, but only two tasks need to be served initially to keep schedulability for this example. The maximal slack vector is determined to be $\hat{\mathbf{s}} = [1, 1, \infty]^\intercal$ and the ∞ shows that a third task is not served in this action. Using this $\hat{\mathbf{s}}$, the initial action was chosen to be tasks $B^{(3)}$ and $B^{(4)}$ in Figure 4.1. However, these tasks are not the only ones that could be chosen to correspond to the maximal slack vector. Tasks $B^{(1)}$ or $B^{(2)}$ could have been chosen to be served in the initial action as well since $s^{(1)} = s^{(2)} = s^{(3)} = s^{(4)} = 1$. Also, it is possible to serve another task, with any slack z, together with the two tasks chosen in this example, as $[1, 1, z]^\intercal \preccurlyeq [1, 1, \infty]^\intercal$ for any z. [1]

4.2.2 Scalar vs. Vector Example

A schedule in Figure 4.2 is shown with the initial action serving $B^{(1)}$, $B^{(4)}$ and $B^{(7)}$ corresponding to a slack vector of $[0, 1, 3]^\intercal$. A gray box indicates that at time 0 the task i is served ($u_k^{(i)} = 1$) while a white box indicates that the task is idle ($u_k^{(i)} = 0$). The shaded boxes show a possible projected schedule for the future providing that the chosen action at time 0 is admissible. Each box also contains the numerical values of $D^{(i)}$. As all the demands are served for all tasks after the iteration at time $k = 3$, the problem is schedulable.

The example in Figure 4.2 shows that a scalar such as the sum of the slacks does not necessarily determine admissibility of an action. A vector is needed to determine admissibility

Tasks	$B^{(1)}$	$B^{(2)}$	$B^{(3)}$	$B^{(4)}$	$B^{(5)}$	$B^{(6)}$
Demand $D^{(i)}$	4	2	2	3	2	1
Deadline $T^{(i)}$	5	3	3	4	5	5
Slack $s^{(i)}$	1	1	1	1	3	4

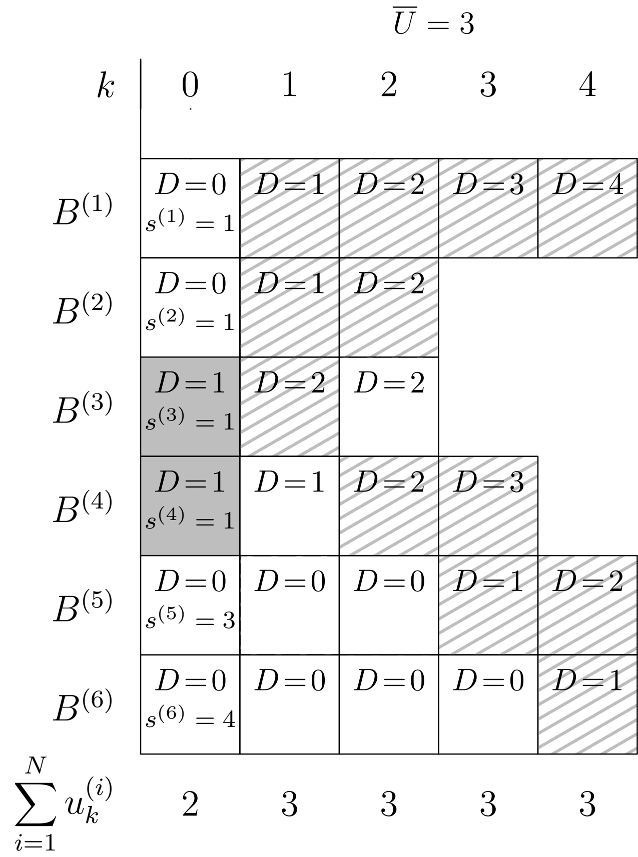

Figure 4.1: Infinity Example Schedule

[1] © 2017 IEEE

Tasks	$B^{(1)}$	$B^{(2)}$	$B^{(3)}$	$B^{(4)}$	$B^{(5)}$	$B^{(6)}$	$B^{(7)}$
Demand $D^{(i)}$	4	1	1	1	2	2	1
Deadline $T^{(i)}$	4	2	2	2	4	4	4
Slack $s^{(i)}$	0	1	1	1	2	2	3

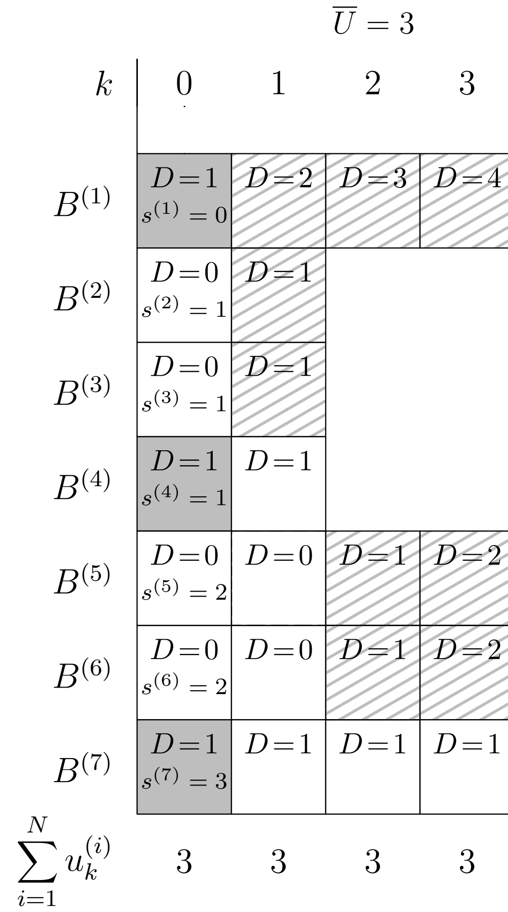

Figure 4.2: Scalar vs. Vector Example

[1] © 2017 IEEE

of an action. This scheduling problem consists of a set of 7 integer tasks with heterogeneous deadlines with $\overline{U} = 3$ where the maximal slack vector is determined to be $\hat{\mathbf{s}} = [0, 1, 3]^\intercal$. Even though $\hat{\mathbf{s}}$ has the same sum as the slack vector $[0, 2, 2]^\intercal$, this slack vector does not correspond to an admissible initial action, and this can be seen in the Figure 4.2. Actions with slack vectors of $[0, 1, 2]^\intercal$ or $[0, 1, 3]^\intercal$ are both admissible because they are both smaller in slack than $\hat{\mathbf{s}}$. [1]

4.2.3 Auction Example

A schedule in Figure 4.3 is shown with the initial action serving $B^{(1)}$, $B^{(3)}$, $B^{(6)}$, and $B^{(7)}$ corresponding to a slack vector of $[0, 1, 3, 4]^\intercal$. A gray box indicates that at time 0 the task i is served ($u_k^{(i)} = 1$) while a white box indicates that the task is idle ($u_k^{(i)} = 0$). The shaded boxes show a possible projected schedule for the future providing that the chosen action at time 0 is admissible. Each box also contains the numerical values of $D^{(i)}$. As all the demands are served for all tasks after the iteration at time $k = 7$, the problem is schedulable.

After determining the maximal slack vector $\hat{\mathbf{s}}$, all of the admissible actions at time zero are known because every action with a slack vector less than $\hat{\mathbf{s}}$ will also be an admissible action. This information can be used to implement an optimal policy when the cost of the current action is available and the cost of future actions is still to be revealed. For example, we could consider a service that guarantees the completion of a task by deadline T. To participate the user must pay an entry fee, but the user may decide to modify its priority and bid in order to be served at specific time slots. A scheme of this kind can be used to implement a demand response mechanism to supply energy to different flexible loads while at the same time take into account their time preferences.

The example in Figure 4.3 is a set of 9 integer tasks with heterogeneous deadlines with $\overline{U} = 4$, and an attempt is made to maximize the profit based on a bidding system assuming no a-priori knowledge of the future bids. By using the Maximal Element Algorithm, the maximal slack vector for this problem at time 0 is computed and $\hat{\mathbf{s}} = [0, 2, 3, 5]^\intercal$. Consequently any action with a smaller slack vector than the maximal slack element is also admissible. In the example, the slack of $B^{(1)}$ is 0 and must be served until it is completed. Therefore, the slack vector $[0, 1, 3, 4]^\intercal$ (corresponding to $B^{(1)}$, $B^{(3)}$, $B^{(6)}$, $B^{(7)}$) would maximize the profit from

Tasks	$B^{(1)}$	$B^{(2)}$	$B^{(3)}$	$B^{(4)}$	$B^{(5)}$	$B^{(6)}$	$B^{(7)}$	$B^{(8)}$	$B^{(9)}$
Demand $D^{(i)}$	4	3	5	2	6	5	3	1	2
Deadline $T^{(i)}$	4	4	6	4	8	8	7	6	8
Slack $s^{(i)}$	0	1	1	2	2	3	4	5	6
Bid $p^{(0)}$	\$1.5	\$2.1	\$5.8	\$1.9	\$2.5	\$3.5	\$3.7	\$2.3	\$7.1

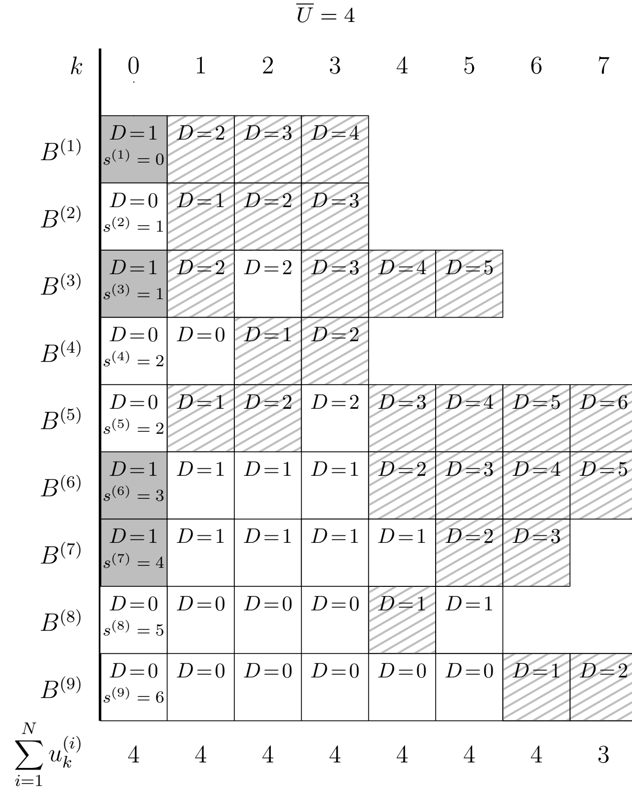

Figure 4.3: Auction Example

[1] © 2017 IEEE

the bids. Even though $B^{(9)}$ has the highest bid, it is not possible to serve this task initially or schedulability for the problem would be lost. [1]

4.3 Incoming Arrivals

One of the next areas that should be studied is the incorporation of incoming arrivals (tasks). To make this algorithm more practical, being able to determine whether new tasks can be added to the system while keeping feasibility is important. In Chapter 2, the Minimum Effort Algorithm was used to test if an action was admissible. This algorithm waits as long as possible to serve a task while still meeting the demands if possible. The advantage to this method is that it is simple to determine if one task can join the system because any remaining capacity (referring to when the $\overline{U}$ is not met) will be available as soon as possible. Then the new demand and deadline of an incoming task can be compared to the remaining capacity quickly to determine whether the incoming task can be added to the system while keeping feasibility. A setback to this method is that while multiple tasks can be quickly tested if they can join the schedule while keeping feasibility, after one task is added to the schedule, the new schedule is not necessarily the new Minimum Effort schedule. Consequently, there may be a scenario when multiple tasks are attempting to join the system where the scheduler does not allow a task to enter, but there may be a different schedule that would allow the task to enter. Attempting to keep the Minimum Effort schedule while adding tasks is one area that could be explored in the future or other methods allowing the addition of tasks to the initial system.

Chapter 5

Proving The Unique Maximal Slack Element Theorem

This chapter contains the definitions, lemmas, and theorems used to come to the conclusion of Theorem 16. [17]

5.1 Common and Difference Tasks

When comparing the tasks that are served in two integer actions, the tasks can be divided into two categories of being common or difference tasks. The following definition is introduced for this comparison of two actions.

Definition 17 (Common and Difference Tasks)**.** Consider two integer actions u_0 and v_0. The set of common tasks between u_0 and v_0 is the vector of tasks

$$\mathcal{C}(u_0, v_0) = (B^{(i_1)}, B^{(i_2)}, ..., B^{(i_n)})$$

such that $s^{(i_1)} \leq s^{(i_2)} \leq ... \leq s^{(i_n)}$ and $B^{(i_k)}$, for $k = 1, ..., n$, are all the n tasks such that $u_0^{(i_k)} = v_0^{(i_k)} = 1$.

The set of difference tasks is u_0 and v_0 is the vector of tasks

$$\overline{\mathcal{C}}(u_0, v_0) = (B^{(i_1)}, B^{(i_2)}, ..., B^{(i_n)})$$

such that $s^{(i_1)} \leq s^{(i_2)} \leq ... \leq s^{(i_n)}$ and $B^{(i_k)}$, for $k = 1, ..., n$, are all the n tasks such that $u_0^{(i_k)} = 1$ and $v_0^{(i_k)} = 0$.

As an example, let action $u_0 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$ and action $v_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$. The common tasks are $\mathcal{C}(u_0, v_0) = (B^{(3)}, B^{(4)})$ under the assumption $s^{(3)} \leq s^{(4)}$. The difference tasks are $\overline{\mathcal{C}}(u_0, v_0)$ $= (B^{(1)})$ and $\overline{\mathcal{C}}(v_0, u_0) = (B^{(5)})$.

5.2 Minimum and Maximum Action

Another definition is introduced.

Definition 18 (Minimum Action)**.** Consider two integer actions u_0 and v_0. Let $\overline{\mathcal{C}}(u_0, v_0)$ $= (B^{(i_1)}, B^{(i_2)}, ..., B^{(i_n)})$ and $\overline{\mathcal{C}}(v_0, u_0) = (B^{(j_1)}, B^{(j_2)}, ..., B^{(j_l)})$. Without loss of generality, assume that $\left\|\overline{\mathcal{C}}(u_0, v_0)\right\|_1 \leq \left\|\overline{\mathcal{C}}(v_0, u_0)\right\|_1$ meaning $n \leq l$. The definition $min(u_0, v_0)$ returns the action m with the following properties:

- $m^{(i)} = 1$ if $B^{(i)} \in \mathcal{C}(u_0, v_0)$
- also for $k = 1, ..., n$
  - $m^{(i_k)} = 1$ and $m^{(j_k)} = 0$ if $slack(B^{(i_k)}) \leq slack(B^{(j_k)})$
  - $m^{(i_k)} = 0$ and $m^{(j_k)} = 1$ if $slack(B^{(i_k)}) > slack(B^{(j_k)})$
- for $k = n, ..., l$
  - $m^{(j_k)} = 1$
- all the other non-specified entries in the action are equal to zero.

In addition, another definition is needed.

Definition 19 (Maximum Action)**.** Consider two integer actions u_0 and v_0. Let $\overline{\mathcal{C}}(u_0, v_0)$ $= (B^{(i_1)}, B^{(i_2)}, ..., B^{(i_n)})$ and $\overline{\mathcal{C}}(v_0, u_0) = (B^{(j_1)}, B^{(j_2)}, ..., B^{(j_l)})$. Without loss of generality, assume that $\left\|\overline{\mathcal{C}}(u_0, v_0)\right\|_1 \leq \left\|\overline{\mathcal{C}}(v_0, u_0\right\|_1$ meaning $n \leq l$. The definition $max(u_0, v_0)$ returns the action M with the following properties:

- $M^{(i)} = 1$ if $B^{(i)} \in \mathcal{C}(u_0, v_0)$

- also for $k = 1, ..., n$
  - $M^{(i_k)} = 0$ and $M^{(j_k)} = 1$ if $slack(B^{(i_k)}) \leq slack(B^{(j_k)})$
  - $M^{(i_k)} = 1$ and $M^{(j_k)} = 0$ if $slack(B^{(i_k)}) > slack(B^{(j_k)})$
- for $k = n, ..., l$
  - $M^{(j_k)} = 0$
- all the other non-specified entries in the action are equal to zero.

Consider the following example for applying the *min* and *max* definition where action $u_0 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$ and action $v_0 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}$. First, one needs to determine the common and difference tasks. The common tasks are $\mathcal{C}(u_0, v_0) = (B^{(3)})$. The difference tasks are $\overline{\mathcal{C}}(u_0, v_0) = (B^{(1)}, B^{(5)})$ and $\overline{\mathcal{C}}(v_0, u_0) = (B^{(2)}, B^{(4)})$. Under the assumption that $s^{(1)} \leq s^{(2)} \leq ... \leq s^{(5)}$, the returned minimum action will be $m = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$ and the returned maximum action will be $M = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}$.

5.3 Two Actions That Differ in One Task

Lemma 20. If two actions differ in only one task meaning $\left\|\overline{\mathcal{C}}(u_0, v_0)\right\|_1 = \left\|\overline{\mathcal{C}}(v_0, u_0)\right\|_1 = 1$ then the two actions are slack-comparable.

Proof. Consider two integer actions u_0 and v_0 where $\overline{\mathcal{C}}(u_0, v_0) = $ (A) and $\overline{\mathcal{C}}(v_0, u_0) = $ (B). With no loss of generality, assume $slack(A) < slack(B)$. The vectors $SlackVector(u_0)$ and $SlackVector(v_0)$ can be organized into four different subvectors. In the first subvector for both slack vectors, only the slacks with $s^{(i)} \leq slack(A)$ will be considered. The second subvector for $SlackVector(u_0)$ will hold $slack(A)$ while the second subvector for $SlackVector(v_0)$ will include the slacks with $slack(A) < s^{(i)} \leq slack(B)$. The third subvector for $SlackVector(u_0)$ will hold the slacks with $slack(A) < s^{(i)} \leq slack(B)$, and the third subvector for $SlackVector(v_0)$ will hold $slack(B)$. The fourth subvector for both slack vectors will be the same with slacks having $s^{(i)} > slack(B)$. In Figure 5.1, the organization

of the slack vectors is shown. Since the two slack vectors are sorted and $SlackVector(u_0)$ is less than $SlackVector(v_0)$ element wise, $SlackVector(u_0) \preccurlyeq SlackVector(v_0)$. $\square$

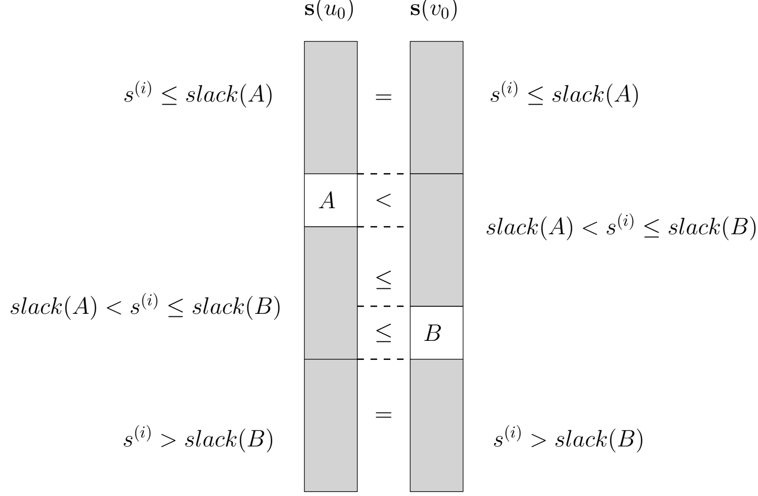

Figure 5.1: Lemma 20 Visualization.

5.4 Corollary to Two Actions Differing in One Task

Lemma 20 can be extended to the following situation.

Corollary 21. Consider two integer actions u_0 and v_0 where $\overline{\mathcal{C}}(u_0, v_0) = (A_1, ..., A_m)$ and $\overline{\mathcal{C}}(v_0, u_0) = (B_1, ..., B_m)$ with the following relations:

$$slack(A_i) \leq slack(A_{i+1}) \text{ for } i = 1, ..., m-1$$
$$slack(B_i) \leq slack(B_{i+1}) \text{ for } i = 1, ..., m-1$$
$$slack(A_i) \leq slack(B_i) \text{ for } i = 1, ..., m.$$

With these relations, $SlackVector(u_0) \preccurlyeq SlackVector(v_0)$ must be true.

Proof. The straight forward proof can be completed with induction on m where the base step $m = 1$ is given by Lemma 20. $\square$

5.5 Actions That Are Not Slack-Comparable

Lemma 22. Consider two integer actions u_0 and v_0 where $\overline{\mathcal{C}}(u_0, v_0)$ = (A,D) and $\overline{\mathcal{C}}(v_0, u_0)$ = (B,C). With no loss of generality, assume $slack(A) \leq slack(D)$ and $slack(B) \leq slack(C)$. u_0 and v_0 are not slack-comparable iff one of the following holds:

$$\textit{i)}\ slack(A) < slack(B)\ \textit{and}\ slack(D) > slack(C)$$

$$\textit{or}$$

$$\textit{ii)}\ slack(A) > slack(B)\ \textit{and}\ slack(D) < slack(C)$$

Proof. "$\Longrightarrow$" This statement will be proven by contradiction. Assume that both i) and ii) do not hold:

$$\text{i}')\ slack(A) \geq slack(B) \text{ or } slack(D) \leq slack(C)$$

$$\text{and}$$

$$\text{ii}')\ slack(A) \leq slack(B) \text{ or } slack(D) \geq slack(C)$$

$Case\,1 : slack(A) \geq slack(B)$ and $slack(A) \leq slack(B)$

Due to the constraints in this case, $slack(A) = slack(B)$ must be true. If a pair of tasks in two actions are different but have the same slack, then either task can be served in one of the actions without affecting the admissibility characteristic of that action (Theorem 15). Therefore, with no loss of generality, $\overline{\mathcal{C}}(u_0, v_0)$ = (D) and $\overline{\mathcal{C}}(v_0, u_0)$ = (C). Using Lemma 20, the actions will be slack-comparable.

$Case\,2 : slack(D) \leq slack(C)$ and $slack(A) \leq slack(B)$

Create a new action, l_0, with the following tasks $\mathcal{C}(u_0, v_0) \bigcup (A, C)$. Hence, $\overline{\mathcal{C}}(u_0, l_0)$ = (D) and $\overline{\mathcal{C}}(l_0, u_0)$ = (C) and using Lemma 20, one can determine $SlackVector(u_0) \preccurlyeq SlackVector(l_0)$. In addition, $\overline{\mathcal{C}}(v_0, l_0)$ = (B) and $\overline{\mathcal{C}}(l_0, v_0)$ = (A) and using Lemma 20, one can determine $SlackVector(l_0) \preccurlyeq SlackVector(v_0)$. With this information, it can be determined that u_0 and v_0 are slack-comparable: $SlackVector(u_0) \preccurlyeq SlackVector(l_0) \preccurlyeq SlackVector(v_0)$.

$Case\,3 : slack(A) \geq slack(B)$ and $slack(D) \geq slack(C)$

Create a new action, l_0, with the following tasks $\mathcal{C}(u_0, v_0) \bigcup (A, C)$. Hence, $\overline{\mathcal{C}}(u_0, l_0)$ = (D) and $\overline{\mathcal{C}}(l_0, u_0)$ = (C) and using Lemma 20, one can determine $SlackVector(l_0) \preccurlyeq$

$SlackVector(u_0)$. In addition, $\overline{\mathcal{C}}(v_0, l_0) = $ (B) and $\overline{\mathcal{C}}(l_0, v_0) = $ (A) and using Lemma 20, it can be determined that $SlackVector(v_0) \preccurlyeq SlackVector(l_0)$. With this information, one can determine u_0 and v_0 are slack-comparable: $SlackVector(v_0) \preccurlyeq SlackVector(l_0) \preccurlyeq SlackVector(u_0)$.

$Case\,4 : slack(D) \leq slack(C)$ and $slack(D) \geq slack(C)$

Due to the constraints in this case, $slack(C) = slack(D)$ must be true. If a pair of tasks in two actions are different but have the same slack, then either task can be served in one of the actions without affecting the admissibility characteristic of that action (Theorem 15). Therefore, with no loss of generality, $\overline{\mathcal{C}}(u_0, v_0) = $ (A) and $\overline{\mathcal{C}}(v_0, u_0) = $ (B). Using Lemma 20, the actions will be slack-comparable.

These four cases show that if both i) and ii) do not hold, then the two actions will be slack-comparable. Therefore, the contrapositive will be true as well: If two actions are not slack-comparable, then either i) or ii) hold (since i) and ii) are mutually exclusive).

□

Proof. "$\Longleftarrow$" One can organize u_0 and v_0 into seven different subvectors to show that these two actions will not be slack-comparable while condition i) holds. The organization for the slack vectors is shown in Figure 5.2, and the Figure shows that while the actions are sorted, neither $SlackVector(v_0) \preccurlyeq SlackVector(u_0)$ or $SlackVector(u_0) \preccurlyeq SlackVector(v_0)$ hold. In a similar way, the actions can be sorted under condition ii) to show that they are not slack-comparable. Consequently, u_0 and v_0 are not slack-comparable while i) or ii) hold. □

5.6 Actions That Are Not Slack-Comparable and Differ in Two Tasks

The following theorem shows that given two admissible actions that are not slack comparable with $\left\|\overline{\mathcal{C}}(u_0, v_0)\right\|_1 = \left\|\overline{\mathcal{C}}(v_0, u_0)\right\|_1 = 2$, another admissible action can be found.

Theorem 23. Consider the scheduling problem with tasks $B^{(i)} = (D^{(i)}, T^{(i)}, 1)$ for $i = 1,...,N$ together with aggregate constraint $\overline{U}$, and let Assumption 8 hold. Let u_0 and v_0 be admissible actions at time $t = 0$ that differ only in two tasks:

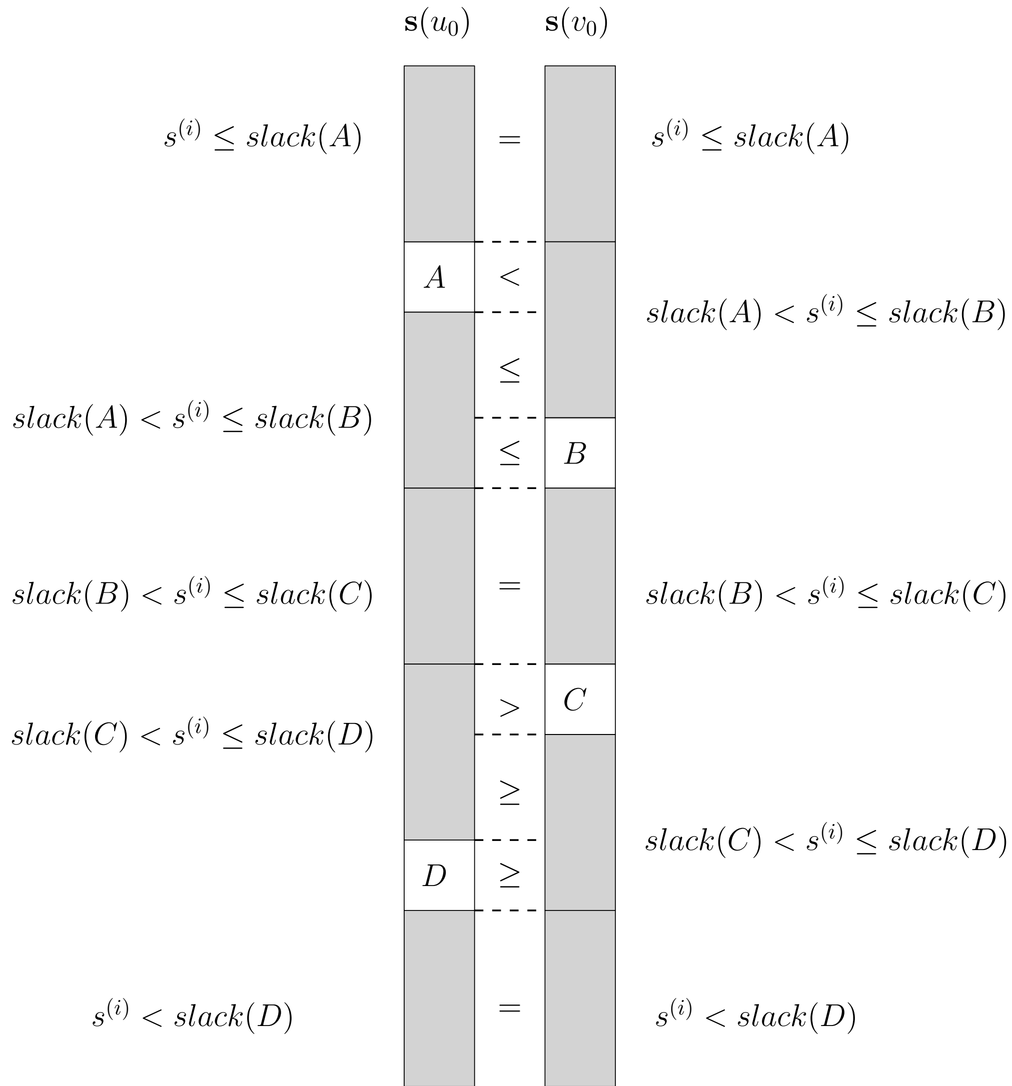

Figure 5.2: Lemma 22 Visualization.

$$u_0 = \overline{\mathcal{C}}(u_0, v_0) \bigcup \mathcal{C}(u_0, v_0)$$
$$v_0 = \overline{\mathcal{C}}(v_0, u_0) \bigcup \mathcal{C}(v_0, u_0)$$

where $\overline{\mathcal{C}}(u_0, v_0) =$ (A,D) and $\overline{\mathcal{C}}(v_0, u_0) =$ (B,C). Assume the slack vectors of the tasks follow these relations:

$$slack(A) < slack(B) \leq slack(C) < slack(D).$$

Then the action:

$$p_0 = \text{(B,D)} \bigcup \mathcal{C}(u_0, v_0)$$

is admissible. In addition, the new action will have the following slack relations:

$$SlackVector(u_0) \preccurlyeq SlackVector(p_0) \quad \text{and} \quad SlackVector(v_0) \preccurlyeq SlackVector(p_0).$$

Proof. The proof is in Appendix A. □

5.7 Maximal Element When Two Actions Serve The Same Number Of Tasks

Lemma 24. Given a schedulable problem, if u_0 and v_0 are maximal actions and $\left\|\overline{\mathcal{C}}(u_0, v_0)\right\|_1 = \left\|\overline{\mathcal{C}}(v_0, u_0)\right\|_1$, then $SlackVector(u_0) = SlackVector(v_0)$.

Proof. This lemma is proven by contradiction. Let u_0 and v_0 be two maximal admissible actions with distinct slack vectors. Since u_0 and v_0 are both maximal and have different slack vectors, they are not slack-comparable. There is no loss of generality if the tasks in $\overline{\mathcal{C}}(u_0, v_0)$ are allowed to have different slacks compared to the tasks in $\overline{\mathcal{C}}(v_0, u_0)$. Indeed, if a pair of tasks in u_0 and v_0 are different but have the same slack, then using Theorem 15 one can replace the task in u_0 with the task in v_0 obtaining a new action that is still admissible. Let $m = min(u_0, v_0)$ and $M = max(u_0, v_0)$.

Observe that any task in $\overline{\mathcal{C}}(u_0, v_0)$ must either be in m or in M. Thus, partition the tasks in $\overline{\mathcal{C}}(u_0, v_0)$ in the following way. $(A_1, ..., A_{n_A})$ are the tasks in $\overline{\mathcal{C}}(u_0, M)$ and $(D_1, ..., D_{n_D})$ are the tasks in $\overline{\mathcal{C}}(u_0, m)$ where there is no loss of generality by assuming $slack(A_i) \leq slack(A_{i+1})$ for $i = 1, ..., n_{A-1}$ and $slack(D_i) \leq slack(D_{i+1})$ for $i = 1, ..., n_{D-1}$.

Observe that any task in $\overline{\mathcal{C}}(v_0, u_0)$ must either be in m or in M. Thus, partition the tasks in $\overline{\mathcal{C}}(v_0, u_0)$ in the following way. $(B_1, ..., B_{n_B})$ are the tasks in $\overline{\mathcal{C}}(v_0, M)$ and $(C_1, ..., C_{n_C})$ are the tasks in $\overline{\mathcal{C}}(v_0, m)$ where there is no loss of generality by assuming $slack(B_i) \leq slack(B_{i+1})$ for $i = 1, ..., n_{B-1}$ and $slack(C_i) \leq slack(C_{i+1})$ for $i = 1, ..., n_{C-1}$.

Since $\left\|\overline{\mathcal{C}}(u_0, v_0)\right\|_1 = \left\|\overline{\mathcal{C}}(v_0, u_0)\right\|_1$, we have that $n_A = n_B$ and $n_C = n_D$. By the construction of m and M, we have that $slack(A_i) < slack(B_i)$ and $slack(C_i) < slack(D_i)$. We have $n_A \geq 1$ and $n_D \geq 1$, otherwise u_0 and v_0 would be slack-comparable, according to Lemma 20.

The minimum action m contains $\mathcal{C}(u_0, v_0) \bigcup (A_1, ..., A_{n_A}) \bigcup (C_1, ..., C_{n_C})$ so $SlackVector(m) \preccurlyeq SlackVector(u_0)$ and $SlackVector(m) \preccurlyeq SlackVector(v_0)$ implying that m is admissible. Follow these iterative steps.

1. Create a new action, u'_0, by taking m and replacing the task C_1 with task D_1. u'_0 contains $\mathcal{C}(u_0, v_0) \bigcup (A_1, ..., A_{n_A}) \bigcup (D_1, C_2, ..., C_{n_C})$. According to Corollary 21, u'_0 will still be admissible because it will be smaller in slack than u_0.

2. Create a new action, v'_0, by taking m and replacing the task A_1 with task B_1. v'_0 contains $\mathcal{C}(u_0, v_0) \bigcup (B_1, A_2, ..., A_{n_A}) \bigcup (C_1, ..., C_{n_C})$. According to Corollary 21, v'_0 will still be admissible because it will be smaller in slack than v_0.

3. Since $\overline{\mathcal{C}}(u'_0, v'_0) = (A_1, D_1)$ and $\overline{\mathcal{C}}(v'_0, u'_0) = (B_1, C_1)$, u'_0 and v'_0 are not slack-comparable according to Lemma 22. Therefore, we can create a new admissible action q_1 according to Theorem 23.

   - q_1 contains $\mathcal{C}(u_0, v_0) \bigcup (B_1, A_2, ..., A_{n_A}) \bigcup (D_1, C_2, ..., C_{n_C})$.

4. Create a new action, u'_1, that contains $\mathcal{C}(u_0, v_0) \bigcup (A_1, ..., A_{n_A}) \bigcup (D_1, D_2, C_3, ..., C_{n_C})$. According to Corollary 21, u'_1 will still be admissible because it will be smaller in slack than u_0.

5. Since $\overline{\mathcal{C}}(u_1', q_1)=(A_1, D_2)$ and $\overline{\mathcal{C}}(q_1, u_1')=(B_1, C_2)$, u_1' and q_1 are not slack-comparable according to Lemma 22. Therefore, we can create a new admissible action q_2 according to Theorem 23.

   - q_2 contains $\mathcal{C}(u_0, v_0) \bigcup (B_1, A_2, ..., A_{n_A}) \bigcup (D_1, D_2, C_3, ..., C_{n_C})$.

6. Iterate steps from 4 to 5 n_C times creating a sequence of actions $q_1, ..., q_{n_C}$ where $q_{n_C} = \mathcal{C}(u_0, v_0) \bigcup (B_1, A_2, ..., A_{n_A}) \bigcup (D_1, D_2, ..., D_{n_D})$.

   - Observe that $\overline{\mathcal{C}}(u_0, q_{n_C}) = (A_1)$ and $\overline{\mathcal{C}}(q_{n_C}, u_0) = (B_1)$.

The actions q_{n_C} and u_0 are slack-comparable according to Lemma 20 providing as a contradiction $SlackVector(u_0) \preccurlyeq SlackVector(q_{n_C})$ but not $SlackVector(q_{n_C}) \preccurlyeq SlackVector(u_0)$.

□

5.8 Proof for Theorem 16

By contradiction assume u_0, v_0 are two maximal actions where $SlackVector(u_0)$ and $SlackVector(v_0)$ are not slack-comparable. If $\left\|\overline{\mathcal{C}}(u_0, v_0)\right\|_1 = \left\|\overline{\mathcal{C}}(v_0, u_0)\right\|_1$, then Lemma 24 gives the assertion.

Therefore, let us consider the case where $\left\|\overline{\mathcal{C}}(u_0, v_0)\right\|_1 \neq \left\|\overline{\mathcal{C}}(v_0, u_0)\right\|_1$. This implies that $\|u_0\|_1 \neq \|v_0\|_1$. With no loss of generality, let $\|u_0\|_1 < \|v_0\|_1$. From the scheduling problem $\mathcal{P}$ let us define a new scheduling problem $\mathcal{P}'$ that has the same aggregate constraint $\overline{U}$, the same tasks $B^{(i)}, ..., B^{(N)}$ of $\mathcal{P}$ and $M = T\overline{U} - \sum_{i=1}^{N} D^{(i)}$ additional "phantom tasks":

$$B^{(N+1)} = B^{(N+2)} = ... = B^{(M)} = (1, T, 1).$$

$\mathcal{P}'$ is still feasible. Indeed, let $\{w_k\}_{k=0}^{T-1}$ be a schedule for $\mathcal{P}$. $\sum_{k=1}^{T} \|w_k\|_1 = \sum_{i=1}^{N} D^{(i)}$. For each w_k such that $\|w_k\|_1 < \overline{U}$ define w_k' as an action serving the same tasks as in w_k and $\overline{U} - \|w_k\|_1$ phantom tasks that have not been served in the interval $[0, K-1]$. The schedule $\{w_k'\}_{k=0}^{T-1}$ is obviously feasible for $\mathcal{P}'$. Also, observe that $\mathcal{P}$ is a relaxation of $\mathcal{P}'$: if $\{w_k'\}_{k=0}^{T-1}$ is a feasible schedule for $\mathcal{P}'$, a feasible schedule $\{w_k\}_{k=0}^{T-1}$ for $\mathcal{P}$ can be obtained by removing from each

action w'_k the phantom tasks. As a consequence the slack vector $SlackVector(w_0)$ is given by replacing in $SlackVector(w'_0)$ the slacks of the phantom tasks (that are all equal to $T-1$) with ∞. Also, observe that each admissible action w'_0 for $\mathcal{P}'$ is such that $\|w'_0\|_1 = \overline{U}$. An example of creating a new schedule with phantom tasks is shown in Figure 5.3. (Figure 5.3 shows a hypothetical schedules for $\mathcal{P}$ and $\mathcal{P}'$. In the schedule for $\mathcal{P}$, the dots represent a task being served while a blank space is a place holder representing that another task could be served at that time step. $\mathcal{P}'$ takes $\mathcal{P}$ and adds M phantom tasks, and the Xs in the schedule for $\mathcal{P}'$ represent phantom tasks being served.)

Since all admissible actions in $\mathcal{P}'$ serve $\overline{U}$ tasks, from Lemma 24 all the maximal admissible actions have the same slack vector. Let $\hat{w}'_0$ be a maximal admissible action for $\mathcal{P}'$ such that the number of phantom tasks served at time zero is maximized as well. At the same time, let u'_0 and v'_0 be the two admissible actions for $\mathcal{P}'$ obtained from u_0 and v_0 in $\mathcal{P}$ and by serving $\overline{U} - \|u_0\|_1$ and $\overline{U} - \|v_0\|_1$ phantom tasks respectively.

Since $\hat{w}'_0$ is maximal we have $SlackVector(\hat{w}'_0) \succcurlyeq SlackVector(u'_0)$ and $SlackVector(\hat{w}'_0) \succcurlyeq SlackVector(v'_0)$. Observe that $\hat{w}'_0$ serves at least as many phantom tasks as u'_0 and strictly more phantom tasks than v'_0 does. Indeed let n_u and n_w be the number of phantom tasks served by u'_0 and w'_0 respectively. If by contradiction, $n_u > n_w$, there is at least one phantom served by u'_0 and not by w'_0. Since $SlackVector(w'_0) \succcurlyeq SlackVector(u'_0)$, there are at least as many tasks with slack equal to T - 1 served by $SlackVector(w'_0)$ as in $SlackVector(u'_0)$. Thus, we can replace one non-phantom task in w'_0 with another phantom task using Theorem 15. This contradicts the fact that w'_0 is a maximal action serving the largest number of phantom tasks.

Define $\hat{w}_0$ as the action obtained from $\hat{w}'_0$ by removing the phantom tasks. $\hat{w}_0$ is feasible for $\mathcal{P}$ and $SlackVector(\hat{w}_0)$ is obtained by replacing the slacks of the phantom tasks with ∞. Now $SlackVector(\hat{w}_0) \succcurlyeq SlackVector(v_0)$ but not $SlackVector(v_0) \succcurlyeq SlackVector(\hat{w}_0)$ because $\hat{w}'_0$ serves more phantom tasks than v'_0, leading to a contradiction since v_0 is a maximal action.

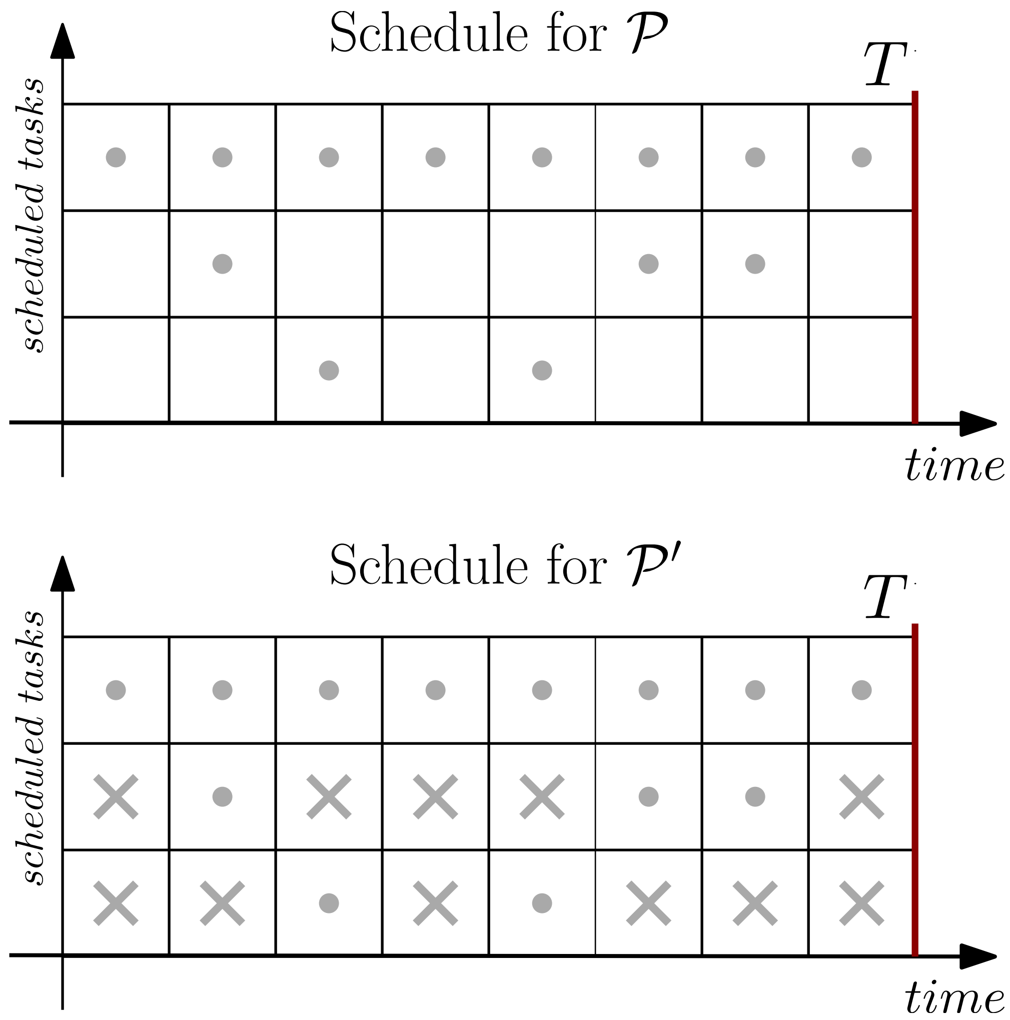

Figure 5.3: Phantom Task Schedule

Chapter 6

Conclusion

For a constrained dynamic scheduling problem, this thesis provides an efficient algorithm to determine all the admissible actions at the current time step. All the admissible actions can be found using a partial ordering approach by finding the maximal slack element. This element can be determined by using the Maximal Element Algorithm and then a scheme can implemented to optimize with respect to the current action instead of the global schedule while keeping feasibility [1]. The algorithm in this thesis can be utilized to create fast algorithms for dynamic scheduling problems in a scenario where future costs are not known a priori. For future work, information about future bids could be incorporated into optimizing the schedule to a cost function. Also, incoming tasks to the system need to be further explored.

Bibliography

[1] S. Haque, D. Materassi, S. Bolognani, M. Roozbehani, and M. A. Dahleh, "An efficient partial-order representation of feasible schedules for online decisions," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, Dec 2017, pp. 2641–2646. iii, iv, 1, 3, 5, 17, 19, 20, 21, 22, 23, 24, 37

[2] M. Pinedo, *Scheduling: theory, algorithms, and systems.* Springer, 2012. 1, 8, 10

[3] A. Zakariazadeh, S. Jadid, and P. Siano, "Multi-objective scheduling of electric vehicles in smart distribution system," *Energy Conversion and Management*, vol. 79, pp. 43 – 53, 2014. 1

[4] P. You, Z. Yang, Y. Zhang, S. H. Low, and Y. Sun, "Optimal charging schedule for a battery switching station serving electric buses," *IEEE Transactions on Power Systems*, vol. 31, no. 5, pp. 3473–3483, 2016. 1

[5] M. Hwang, D. Choi, and P. Kim, "Least slack time rate first: New scheduling algorithm for multi-processor environment," in *2010 International Conference on Complex, Intelligent and Software Intensive Systems*, Feb 2010, pp. 806–811. 2

[6] P. P. Varaiya, F. F. Wu, and J. W. Bialek, "Smart operation of smart grid: Risk-limiting dispatch," *Proceedings of the IEEE*, vol. 99, no. 1, pp. 40–57, 2011. 2

[7] A. Subramanian, M. Garcia, D. Callaway, K. Poolla, and P. Varaiya, "Real-time scheduling of distributed resources," *IEEE Transactions on Smart Grid*, vol. 4, no. 4, pp. 2122–2130, Dec. 2013. 2

[8] M. Kefayati and C. Caramanis, "Efficient energy delivery management for phevs," in *2010 First IEEE International Conference on Smart Grid Communications.* IEEE, 2010, pp. 525–530. 3

[9] E. Bitar and Y. Xu, "Deadline differentiated pricing of deferrable electric loads," *IEEE Transactions on Smart Grid*, vol. 8, no. 1, pp. 13–25, 2017. 3

[10] H. Hao and W. Chen, "Characterizing flexibility of an aggregation of deferrable loads," in *IEEE Conference on Decision and Control.* IEEE, 2014, pp. 4059–4064. 3

[11] G. Gupta, V. K. Kumawat, P. R. Laxmi, D. Singh, V. Jain, and R. Singh, "A simulation of priority based earliest deadline first scheduling for cloud computing system," in *2014 First International Conference on Networks Soft Computing (ICNSC2014)*, Aug 2014, pp. 35–39. 3

[12] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, no. 4, pp. 35:1–35:44, Oct. 2011. 3

[13] M. Bertogna, M. Cirinei, and G. Lipari, "Schedulability analysis of global scheduling algorithms on multiprocessor platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 4, pp. 553–566, 2009. 3

[14] B. Andersson, K. Bletsas, and S. Baruah, "Scheduling arbitrary-deadline sporadic task systems on multiprocessors," in *2008 Real-Time Systems Symposium*, Nov 2008, pp. 385–394. 3

[15] Y.-J. Chen, "Optimal dynamic auctions for display advertising," *Operations Research*, vol. 65, no. 4, pp. 897–913, 2017. 4

[16] A. Hojjat, J. Turner, S. Cetintas, and J. Yang, "A unified framework for the scheduling of guaranteed targeted display advertising under reach and frequency requirements," *Operations Research*, vol. 65, no. 2, pp. 289–313, 2017. 4

[17] S. Haque, D. Materassi, S. Bolognani, M. Roozbehani, and M. A. Dahleh, "Title to be decided," Manuscript to Journal of Scheduling in Preparation. 4, 25

[18] D. Materassi, S. Bolognani, M. Roozbehani, and M. Dahleh, "Deferrable loads in an energy market: coordination under congestion constraints," in *Mediterranean Conference on Control and Automation*. IEEE, 2014, pp. 628–633. 5, 9, 12, 13

[19] J. Chen, F. N. Lee, A. M. Breipohl, and R. Adapa, "Scheduling direct load control to minimize system opoperation cost," *IEEE Transactions on Power Systems*, vol. 10, no. 4, pp. 1994–2001, Nov. 1995. 8

[20] K.-H. Ng and G. B. Sheble, "Direct load control - a profit-based load management using linear programming," *IEEE Transactions on Power Systems*, vol. 13, no. 2, pp. 688–695, May 1998. 8

[21] A. J. Hoffman and J. B. Kruskal, *Integral Boundary Points of Convex Polyhedra.* Princeton University Press, 1956, pp. 223–246. 8, 9

Appendix

A Proving Theorem 23

The following theorem shows that given two admissible actions that are not slack comparable with $\|\overline{\mathcal{C}}(u_0, v_0)\| = \|\overline{\mathcal{C}}(v_0, u_0)\| = 2$, another admissible action can be found.

Consider the scheduling problem with tasks $B^{(i)} = (D^{(i)}, T^{(i)}, 1)$ for $i = 1,...,N$ together with aggregate constraint $\overline{U}$, and let Assumption 8 hold. Let u_0 and v_0 be admissible actions at time $t = 0$ that differ only in two tasks:

$$u_0 = \overline{\mathcal{C}}(u_0, v_0) \bigcup \mathcal{C}(u_0, v_0)$$
$$v_0 = \overline{\mathcal{C}}(v_0, u_0) \bigcup \mathcal{C}(v_0, u_0)$$

where $\overline{\mathcal{C}}(u_0, v_0) = (A, D)$ and $\overline{\mathcal{C}}(v_0, u_0) = (B, C)$. Assume the slack vectors of the tasks follow these relations:

$$slack(A) < slack(B) \leq slack(C) < slack(D).$$

Then the action

$$p_0 = (B, D) \bigcup \mathcal{C}(u_0, v_0)$$

is admissible. In addition, the new action will have the following slack relations:

$$SlackVector(u_0) \preccurlyeq SlackVector(p_0) \quad \text{and} \quad SlackVector(v_0) \preccurlyeq SlackVector(p_0).$$

Proof:

Consider the following scheduling problem with two schedules: $\mathcal{U}$ and $\mathcal{V}$.

At a generic time τ, let u_τ and v_τ be the sets of tasks (actions) that are served in schedule $\mathcal{U}$ and in schedule $\mathcal{V}$, respectively.

In the $\mathcal{U}$ schedule, at time 0, we have that

$$u_0 = (A, D) \bigcup \mathcal{C}(u_0, v_0).$$

At time 0, the slacks of the tasks A, B, C, D satisfy

$$slack(A) < slack(B) \leq slack(C) < slack(D).$$

In the time steps after time 0, both schedules are generated by a policy that follows the same common admissible actions for as long as possible. Let t be the time step at which the two schedules need to differ, i.e. there is no action which is an admissible action for both the schedules at time t. This event necessarily needs to happen because the two schedules are both feasible, but $u_0 \neq v_0$. At time t the action chosen for each schedule will be the least slack action.

$$\begin{aligned} \tau = 0: &\qquad \overline{\mathcal{C}}(u_0, v_0) = (A, D) \text{ and } \overline{\mathcal{C}}(v_0, u_0) = (B, C) \\ 1 \leq \tau \leq t-1: &\qquad u_\tau = v_\tau \\ \tau = t: &\qquad u_\tau \neq v_\tau \end{aligned}$$

Notice that at time t, the following relations are true.

$$slack_u(X, t) = slack_v(X, t) \text{ for any task } X \text{ different from } A, B, C, D$$

$$\begin{aligned} slack_u(A, t) &= slack_v(A, t) + 1 &\qquad slack_u(A, t) &> slack_v(A, t) \\ slack_u(B, t) &= slack_v(B, t) - 1 &\qquad slack_u(B, t) &< slack_v(B, t) \\ slack_u(C, t) &= slack_v(C, t) - 1 &\qquad slack_u(C, t) &< slack_v(C, t) \\ slack_u(D, t) &= slack_v(D, t) + 1 &\qquad slack_u(D, t) &> slack_v(D, t) \end{aligned}$$

Using these slack relations and the fact that the action chosen at time t will be the least slack, the following statements must be true.

$$\begin{aligned} &\text{If } A \in u_t, \text{ then } A \in v_t. \\ &\text{If } B \in v_t, \text{ then } B \in u_t. \\ &\text{If } C \in v_t, \text{ then } C \in u_t. \\ &\text{If } D \in u_t, \text{ then } D \in v_t. \end{aligned}$$

At time t, it is known that tasks B, C cannot be completed in schedule $\mathcal{U}$ and that tasks A, D cannot be completed in schedule $\mathcal{V}$. Using this information, it is known that the deadlines of tasks A, B, C, D have not yet passed.

$$T_A, T_B, T_C, T_D > t$$

The goal is to find a feasible schedule $\mathcal{P}$ that has an initial action at time 0 such that

$$p_0 = (B, D) \bigcup \mathcal{C}(u_0, v_0).$$

Remark. If the schedule $\mathcal{P}$ with $p_0 = (C, D) \bigcup \mathcal{C}(u_0, v_0)$ is feasible then there exists a schedule with the initial action $(B, D) \bigcup \mathcal{C}(u_0, v_0)$.

In some cases the schedules can be easily rearranged to prove that $(B, D) \bigcup \mathcal{C}(u_0, v_0)$ is an admissible action at time 0. These are the cases (referred to as Scenario I) in which

- $B \in u_t$ and $A \notin u_t$, or
- $C \in u_t$ and $A \notin u_t$, or
- $D \in v_t$ and $B \notin v_t$, or
- $D \in v_t$ and $C \notin v_t$.

Another way to write the conditions for the cases in Scenario I is

$$(B \in u_t \text{ or } C \in u_t) \text{ and } A \notin u_t$$
$$\text{or}$$
$$(B \notin v_t \text{ or } C \notin v_t) \text{ and } D \in v_t$$

At time t, either $\left\|\overline{\mathcal{C}}(u_t, v_t)\right\| = \left\|\overline{\mathcal{C}}(v_t, u_t)\right\| = 1$ or $\left\|\overline{\mathcal{C}}(u_t, v_t)\right\| = \left\|\overline{\mathcal{C}}(v_t, u_t)\right\| = 2$ must be true.

All the cases for the differing number of elements are considered and the following notation is introduced

$$u_t = u'_t \cup q_t \qquad \text{and} \qquad v_t = v'_t \cup q_t$$

where $u'_t \cap v'_t \subseteq \{A, B, C, D\}$ and $q_t \cap \{A, B, C, D\} = \emptyset$.
It is important to note that $\|u'_t\| = \|v'_t\|$ and that $\|u'_t\| = 1, 2, 3$ or 4.

Therefore, the only cases (referred to as Scenario II) that need to be considered and proven are those where

- $B \notin u_t$ or $A \in u_t$, and

- $C \notin u_t$ or $A \in u_t$, and
- $D \notin v_t$ or $B \in v_t$, and
- $D \notin v_t$ or $C \in v_t$.

Another way to write the cases for Scenario II is

- ($B \notin u_t$ and $C \notin u_t$) and ($B \in v_t$ and $C \in v_t$) or
- $A \in u_t$ and ($B \in v_t$ and $C \in v_t$) or
- ($B \notin u_t$ and $C \notin u_t$) and $D \notin v_t$ or
- $A \in u_t$ and $D \notin v_t$

Cases in which $\|u'_t\| = \|v'_t\| = 1$

Case 1: $u'_t = \{A\}$, $v'_t = \{B\}$ *or* $\{C\}$ *or* $\{X\}$

- Based on the problem configuration it has been determined if $A \in u_t$, then $A \in v_t$. However, we have a contradiction in this case that $A \notin v_t$ and therefore is not possible.

Case 2: $u'_t = \{X\}$, $v'_t = \{B\}$

- Based on the problem configuration it has been determined if $B \in v_t$, then $B \in u_t$. However, we have a contradiction in this case that $B \notin u_t$ and therefore is not possible.

Case 3: $u'_t = \{X\}$, $v'_t = \{C\}$

- Based on the problem configuration it has been determined if $C \in v_t$, then $C \in u_t$. However, we have a contradiction in this case that $C \notin u_t$ and therefore is not possible.

Case 4: $u'_t = \{D\}$, $v'_t = \{A\}$ *or* $\{B\}$ *or* $\{C\}$ *or* $\{X\}$

- Based on the problem configuration it has been determined if $D \in u_t$, then $D \in v_t$. However, we have a contradiction in this case that $D \notin v_t$ and therefore is not possible.

Case 5: $u'_t = \{X\}$, $v'_t = \{A\}$

- The slacks of tasks $slack_u(X,t), slack_v(X,t), slack_u(A,t), slack_v(A,t)$ exist and are defined because tasks A and X are not completed by time t.
- If $slack_u(X,t) \geq slack_u(A,t)$, then could serve A in $u'_t = \{A\}$ meaning the schedules would be the same for u_t and v_t. Based on the problem setup, the schedule has to be different at time t. Therefore, $slack_u(X,t) < slack_u(A,t)$ must be true.
- If $slack_v(A,t) \geq slack_v(X,t)$, then could serve X in $v'_t = \{X\}$ meaning the schedules would be the same for u_t and v_t. Based on the problem setup, the schedule has to be different at time t. Therefore, $slack_v(A,t) < slack_v(X,t)$ must be true.
- Thus using these properties and the initial relations we can determine the following relation

$$slack_v(A,t) < slack_v(X,t) = slack_u(X,t) < slack_u(A,t) = slack_v(A,t) + 1$$

  If this relation holds, $slack_v(X,t)$ and $slack_u(X,t)$ could not be integer. However, all the slacks must be integer based on the problem set up. Therefore, this case is not possible.

Cases in which $\|u'_t\| = \|v'_t\| = 2$

Case 6: $u'_t = \{A,B\}$ *or* $\{A,C\}$ *or* $\{A,D\}$ *or* $\{A,X\}$, $v'_t = \{B,C\}$ *or* $\{B,X\}$ *or* $\{C,X\}$

- Based on the problem configuration it has been determined if $A \in u_t$, then $A \in v_t$. However, we have a contradiction in this case and it is not possible.

Case 7: $u'_t = \{A,C\}$ *or* $\{A,D\}$ *or* $\{A,X\}$ *or* $\{D,X\}$, $v'_t = \{A,B\}$ *or* $\{B,C\}$ *or* $\{B,X\}$

- Based on the problem configuration it has been determined if $B \in v_t$, then $B \in u_t$. However, we have a contradiction in this case and it is not possible.

Case 8: $u'_t = \{A,B\}$ *or* $\{A,D\}$ *or* $\{A,X\}$ *or* $\{D,X\}$, $v'_t = \{A,C\}$ *or* $\{C,X\}$

- Based on the problem configuration it has been determined if $C \in v_t$, then $C \in u_t$. However, we have a contradiction in this case and it is not possible.

Case 9: $u'_t = \{A, D\}$ *or* $\{D, X\}$, $v'_t = \{A, X\}$

- Based on the problem configuration it has been determined if $D \in u_t$, then $D \in v_t$. However, we have a contradiction in this case and it is not possible.

Case 10: $u'_t = \{A, B\}$, $v'_t = \{A, X\}$

- Since task A has been served once more in schedule $\mathcal{U}$ compared to schedule $\mathcal{V}$ at time t, $\exists\, \hat{t} : t < \hat{t} < T_A$ such that $A \notin u_{\hat{t}}$.

- Denote a task as Z that is in $u_{\hat{t}}$ but not in u_t. This task Z must exist because task A is in u_t but not in $u_{\hat{t}}$ meaning there is at least one task different between actions u_t and $u_{\hat{t}}$.

- First, a task Z is moved from $u_{\hat{t}}$ to replace B in u_t. Then move task B from u_t to replace task A in u_0 and finally move A from u_0 to $u_{\hat{t}}$ to obtain a feasible schedule that meets the goal.

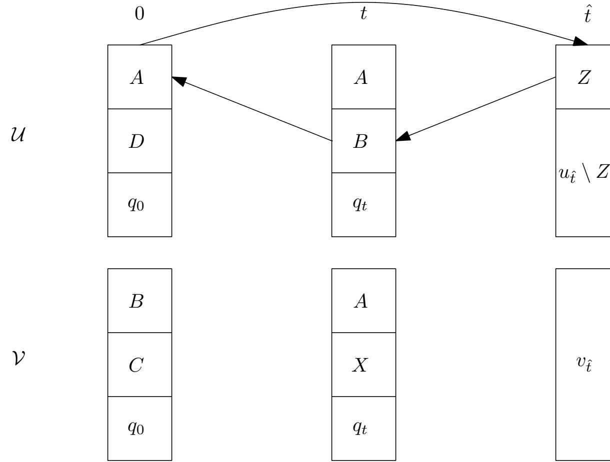

Figure A.1: Case 10 Visualization.

Case 11: $u'_t = \{A, C\}$, $v'_t = \{A, X\}$

- Since task A has been served once more in schedule $\mathcal{U}$ compared to schedule $\mathcal{V}$ at time t, $\exists\, \hat{t} : t < \hat{t} < T_A$ such that $A \notin u_{\hat{t}}$.

- Denote a task as Z that is in $u_{\hat{t}}$ but not in u_t. This task Z must exist because task A is in u_t but not in $u_{\hat{t}}$ meaning there is at least one task different between actions u_t and $u_{\hat{t}}$.
- First, a task Z is moved from $u_{\hat{t}}$ to replace C in u_t. Then move task C from u_t to replace task A in u_0 and finally move A from u_0 to $u_{\hat{t}}$ to obtain a feasible schedule that meets the goal.

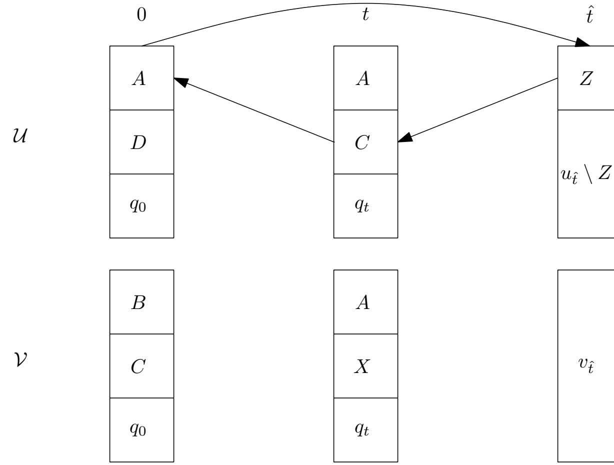

Figure A.2: Case 11 Visualization.

Cases in which $\|u_t'\| = \|v_t'\| = 3$

Case 12: $\boldsymbol{u_t' = \{A,B,C\}}$ ***or*** $\boldsymbol{\{A,B,D\}}$ ***or*** $\boldsymbol{\{A,B,X\}}$ ***or*** $\boldsymbol{\{A,C,D\}}$ ***or*** $\boldsymbol{\{A,C,X\}}$ ***or*** $\boldsymbol{\{A,D,X\}}$, $\boldsymbol{v_t' = \{B,C,D\}}$ ***or*** $\boldsymbol{\{B,C,X\}}$

- Based on the problem configuration it has been determined if $A \in u_t$, then $A \in v_t$. However, we have a contradiction in this case and it is not possible.

Case 13: $\boldsymbol{u_t' = \{A,C,D\}}$ ***or*** $\boldsymbol{\{A,C,X\}}$ ***or*** $\boldsymbol{\{A,D,X\}}$, $\boldsymbol{v_t' = \{A,B,C\}}$ ***or*** $\boldsymbol{\{A,B,X\}}$

- Based on the problem configuration it has been determined if $B \in v_t$, then $B \in u_t$. However, we have a contradiction in this case and it is not possible.

Case 14: $\boldsymbol{u_t' = \{A,B,D\}}$ ***or*** $\boldsymbol{\{A,B,X\}}$ ***or*** $\boldsymbol{\{A,D,X\}}$, $\boldsymbol{v_t' = \{A,B,C\}}$ ***or*** $\boldsymbol{\{A,C,X\}}$

- Based on the problem configuration it has been determined if $C \in v_t$, then $C \in u_t$. However, we have a contradiction in this case and it is not possible.

Case 15: $u'_t = \{A, B, D\}$ *or* $\{A, C, D\}$, $v'_t = \{A, B, X\}$ *or* $\{A, C, X\}$

- Based on the problem configuration it has been determined if $D \in u_t$, then $D \in v_t$. However, we have a contradiction in this case and it is not possible.

Case 16: $u'_t = \{A, B, C\}$, $v'_t = \{A, B, X\}$

- Since task A has been served once more in schedule $\mathcal{U}$ compared to schedule $\mathcal{V}$ at time t, $\exists\, \hat{t} : t < \hat{t} < T_A$ such that $A \notin u_{\hat{t}}$.

- Denote a task as Z that is in $u_{\hat{t}}$ but not in u_t. This task Z must exist because task A is in u_t but not in $u_{\hat{t}}$ meaning there is at least one task different between actions u_t and $u_{\hat{t}}$.

- First, a task Z is moved from $u_{\hat{t}}$ to replace B in u_t. Then move task B from u_t to replace task A in u_0 and finally move A from u_0 to $u_{\hat{t}}$ to obtain a feasible schedule that meets the goal.

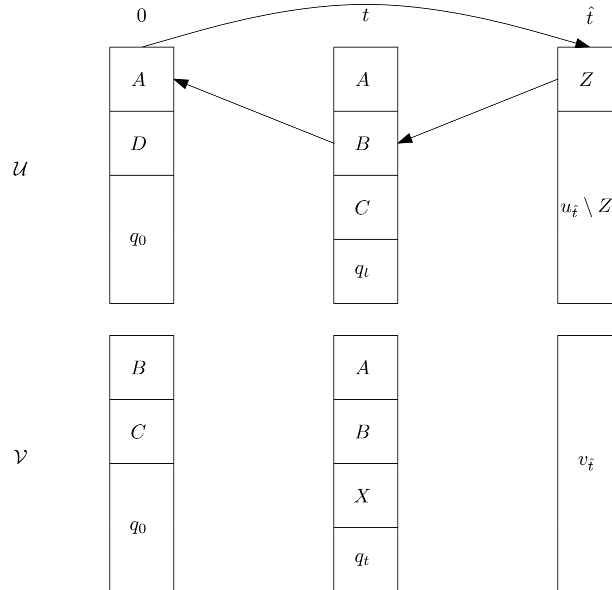

Figure A.3: Case 16 Visualization.

Case 17: $u'_t = \{A, B, C\}$, $v'_t = \{A, C, X\}$

- Since task A has been served once more in schedule $\mathcal{U}$ compared to schedule $\mathcal{V}$ at time t, $\exists\, \hat{t} : t < \hat{t} < T_A$ such that $A \notin u_{\hat{t}}$.

- Denote a task as Z that is in $u_{\hat{t}}$ but not in u_t. This task Z must exist because task A is in u_t but not in $u_{\hat{t}}$ meaning there is at least one task different between actions u_t and $u_{\hat{t}}$.

- First, a task Z is moved from $u_{\hat{t}}$ to replace C in u_t. Then move task C from u_t to replace task A in u_0 and finally move A from u_0 to $u_{\hat{t}}$ to obtain a feasible schedule that meets the goal.

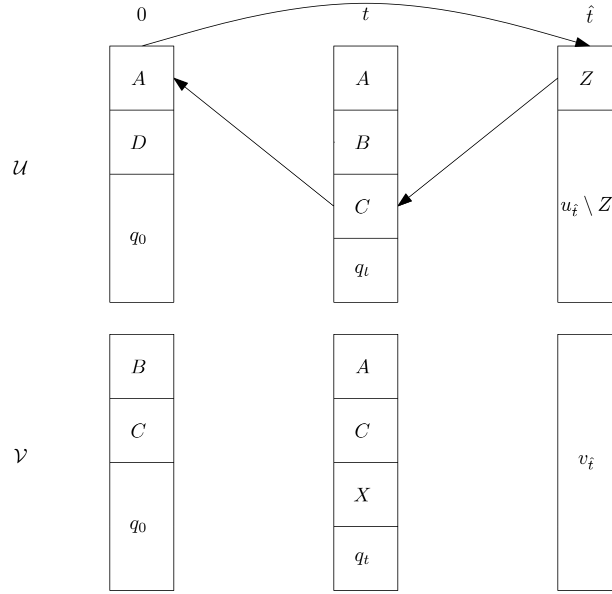

Figure A.4: Case 17 Visualization.

Cases in which $\|u'_t\| = \|v'_t\| = 4$

Case 18: $\boldsymbol{u'_t = \{A, B, C, D\}}$ ***or*** $\boldsymbol{\{A, B, C, X\}}$ ***or*** $\boldsymbol{\{A, B, D, X\}}$ ***or*** $\boldsymbol{\{A, C, D, X\}}$**,**
$\boldsymbol{v'_t = \{B, C, D, X\}}$

- Based on the problem configuration it has been determined if $A \in u_t$, then $A \in v_t$. However, we have a contradiction in this case and it is not possible.

Case 19: $\boldsymbol{u'_t = \{A, C, D, X\}}$ **,** $\boldsymbol{v'_t = \{A, B, C, D\}}$ ***or*** $\boldsymbol{\{A, B, C, X\}}$

- Based on the problem configuration it has been determined if $B \in v_t$, then $B \in u_t$. However, we have a contradiction in this case and it is not possible.

Case 20: $\boldsymbol{u'_t = \{A, B, D, X\}}$ **,** $\boldsymbol{v'_t = \{A, B, C, D\}}$ ***or*** $\boldsymbol{\{A, B, C, X\}}$

- Based on the problem configuration it has been determined if $C \in v_t$, then $C \in u_t$. However, we have a contradiction in this case and it is not possible.

Case 21: $u'_t = \{A, B, C, D\}$, $v'_t = \{A, B, C, X\}$

- Based on the problem configuration it has been determined if $D \in u_t$, then $D \in v_t$. However, we have a contradiction in this case and it is not possible.

Case 22: $u'_t = \{A, B, C, X\}$, $v'_t = \{A, B, C, D\}$

- Since task A has been served once more in schedule $\mathcal{U}$ compared to schedule $\mathcal{V}$ at time t, $\exists\, \hat{t} : t < \hat{t} < T_A$ such that $A \notin u_{\hat{t}}$.

- Denote a task as Z that is in $u_{\hat{t}}$ but not in u_t. This task Z must exist because task A is in u_t but not in $u_{\hat{t}}$ meaning there is at least one task different between actions u_t and $u_{\hat{t}}$.

- First, a task Z is moved from $u_{\hat{t}}$ to replace B in u_t. Then move task B from u_t to replace task A in u_0 and finally move A from u_0 to $u_{\hat{t}}$ to obtain a feasible schedule that meets the goal.

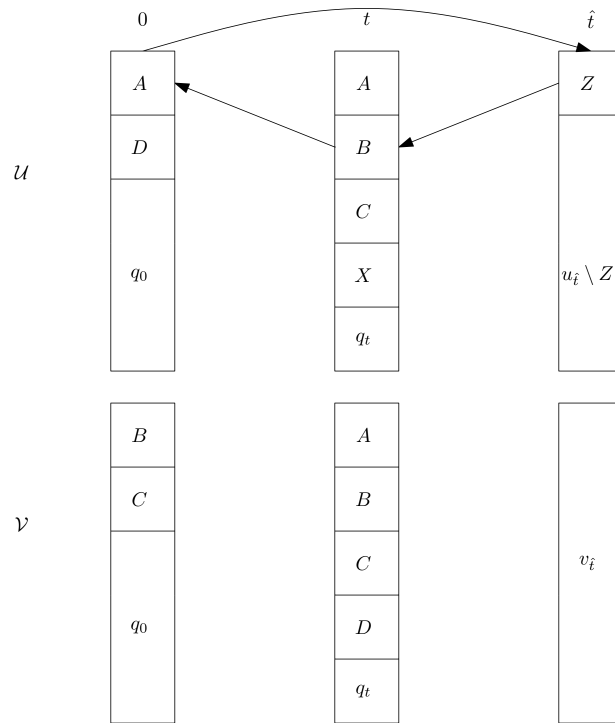

Figure A.5: Case 22 Visualization.

Vita

Saajid Haque was born in Oak Ridge, Tennessee and graduated from Oak Ridge High School in 2012. After gaining an interest in math and science in high school, he decided to pursue a degree in Electrical Engineering at the University of Tennessee, Knoxville. In May 2016, he graduated with a Bachelors of Science in Electrical Engineering from the university and decided to continue his education through graduate school. He plans to officially graduate in August 2018 with a Master of Science in Electrical Engineering with a Concentration in Control Systems. Supplementing his school and research work, Saajid has interned with Oak Ridge National Laboratory, Alcoa, and AT&T.