Applied Mathematics
Faculty of EEMCS
Chair MSCT

UNIVERSITY OF TWENTE.

Optimal Dynamic Voltage and Frequency Scaling for Multimedia Devices

Master thesis (FINAL DRAFT)
August 22, 2011

Supervisor:
dr. J.W. Polderman

Committee:
dr. J.W. Polderman
prof.dr. A.A. Stoorvogel
dr.ir. J. Kuper

Marco Gerards
0124699
m.e.t.gerards@utwente.nl

Contents

Definitions

$\mathbb{R}^+$	The set of positive numbers $\mathbb{R}^+ := \{x \in \mathbb{R} \| x > 0\}$
$\mathbb{R}_0^+$	The set of nonnegative numbers $\mathbb{R}_0^+ := \{x \in \mathbb{R} \| x \geq 0\}$
$\min\limits_{x \in S} f(x)$	The smallest value attained by the function f on the set S
$\max\limits_{x \in S} f(x)$	The biggest value attained by the function f on the set S
$\arg\max\limits_{x \in S} f(x)$	Set defined by $\arg\max\limits_{x \in S} f(x) := \{\bar{x} \in S \| f(\bar{x}) = \max\limits_{x \in S} f(x)\}$
MB	Megabyte; 1,048,576 bytes
GB	Gigabyte; 1,024MB

Nomenclature

$\alpha : [0,t] \to A$	scaling function, page 7
$\bar{p} : [L,1] \to [0,\infty]$	energy per work ($\bar{p}(\alpha) = \frac{p(\alpha)}{\alpha}$), page 8
$\Delta d_i \in \mathbb{R}^+$	time difference between the deadline of task i and task $i+1$, page 9
$\tilde{w}$	Predicted work, page 52
A	Set of scaling factors, either $A = [L,1]$ or $A = \{\bar{\alpha}_1, \ldots, \bar{\alpha}_M\}$, page 7
$b \in \mathbb{R}_0^+$	start time, page 8
$D \in \mathbb{N}$	time difference between the deadline of task i and $i+1$ if the process has period 1, page 10
$d \in \mathbb{R}^+$	deadline, page 7
$f \in \mathbb{R}^+$	completion time, page 8
$L \in \mathbb{R}^+$	lower bound of the scaling factor, page 7
$M \in \mathbb{N}$	number of scaling factors, page 9
$N \in \mathbb{N}$	number of tasks, page 8
$p : [L,1] \to [0,\infty)$	power function (convex), page 8
$R_m \in \mathbb{R}^+$	Total work, page 43
$r_{n,m} \in \mathbb{R}^+$	fraction of work of task n for scaling factor m, page 9
$s_n \in \mathbb{R}_0^+$	slack time, page 16
$T \in \mathbb{N}$	period of a periodic process, page 10
$t \in \mathbb{R}^+$	execution time, page 7
$W \in \mathbb{R}^+$	Worst Case Work (WCW), page 16
$w \in \mathbb{R}^+$	work, page 7
$w^{(0)} \in \mathbb{R}^+$	work per time at the highest speed, page 8

Abstract

Embedded systems like cellular phones, portable media players, etc. are often used for multimedia and high speed communication applications. The complexity of these applications increases rapidly. Because of this, faster devices are required, while the capacity of batteries does not increase at the same pace. Therefore it is important to make the devices energy efficient. Many multimedia and communication applications are real-time applications. They consist of many tasks that all have a deadline. In a video decoder, decoding a single frame can be seen as such a task. The deadlines are given by the time instants at which the frames have to be displayed. Real-time applications are not allowed to miss their deadlines. This means that a task has to finish all its work before its deadline. The amount of work often fluctuates, but is bounded from above by some constant W. Many applications are designed such that all deadlines are met, even if the work for each task is W. This makes it possible to decrease the speed at which certain tasks are processed and still meet all deadlines. The energy consumption will be reduced with the clock frequency.

In this thesis two problems are considered. The first problem is offline energy minimisation. It is assumed that the work of each task is known before the application is executed. A mathematical model is given in which deadlines are written as constraints and energy consumption as a cost function. This model implies an infinite-dimensional convex optimisation problem. The infinite-dimensional problem is then reduced to an equivalent finite-dimensional problem. By finding the global minimiser, the energy used by running this application can be significantly reduced. This thesis explains how to find a global minimum, even in the case when only a finite set of speeds is available. The Karush Kuhn Tucker conditions are used to achieve this.

The second problem studied in this thesis is the online optimisation problem. All work before the current task is known, for the future tasks only an upper bound of the work and predictions of the work are known. The speed for the next task is determined analytically, this is a result that cannot be found in the literature. The task is executed and the procedure is repeated for following task.

In this thesis, these results are not only given, but also compared to approaches in the literature. The energy per work is an important and useful quantity, but is rarely studied in the literature. Using this quantity, energy inefficient speeds are eliminated and a wider range of realistic problems can be solved using the theory that is presented in this thesis.

The results of offline and online optimisation are evaluated using a video decoder that decodes DVDs. These results are compared to a straightforward (greedy) online approach. Video fragments of 30 minutes were used for testing. It turns out that for playback of these video sequences, the speed has to be changed only a few times. In case only a finite set of speeds is available, the size is an upper bound to the minimum number of times the speed has to be changed. When online energy optimisation is used with perfect predictions (i.e., the actual values), the energy consumption is almost as low as with offline energy minimisation. This gives a theoretical lower bound to online energy minimisation. Furthermore, energy can be minimised given predictions of the work.

Chapter 1

Introduction

1.1 Energy consumption in computing systems

Embedded devices like cellular phones, MP3 players, DVD players, navigation systems, hard disk recorders, gaming devices etc. are becoming increasingly popular. Many of these devices are portable and battery powered. Since embedded devices are becoming more complex, the energy demand of these devices is increasing. The development of batteries of higher capacity cannot keep up with the development of modern processors. New technology has to be developed to decrease the energy consumption of embedded devices and increase the capacity of batteries.

Not only for embedded devices it is important be energy efficient. But also for servers in datacenters it is important to be energy efficient. One of the issues when designing a datacenter is taking care of power dissipation. Google, for instance, builds datacenters close to power plants, but also close to the sea so that sea water can be used for cooling.

The energy consumption per time unit of an (embedded) computer can often be reduced by decreasing the speed of the processor. When there are timing constraints, it is not possible to freely decrease the speed of the device. In a cellular phone, it is not possible to postpone communication arbitrarily. It would not meet the quality standards. The same is true for MP3 players. If the device does not produce audio in time, the user hears clicks. There are a lot of peripherals that can trade time for energy, like computer processors, hard disks, communication networks, etc. In the examples given in this thesis, a computer processor running a video decoder is considered. The results can also be applied to other devices and applications.

1.2 Model of a computer processor

A computer processor is a device that executes a sequence of instructions. The instructions are read from a device called Random Access Memory (RAM). The instruction can instruct the processor to, for instance, read/write data from/to RAM, calculate, compare data, etc. These instructions are very elementary, a typical line of Java, C or Matlab code can result in the execution of many, possibly thousands, instructions. Computer processors operate in discrete time. One time instant is called a *clock cycle*, the number of clock cycles per second is called the *clock frequency*. The actual number of clock cycles that is required to perform a certain calculation depends mainly on the number of instructions and the type of instructions.

It is assumed in this thesis that if the clock frequency is multiplied by a certain factor, the time it takes to execute has to be divided by that factor. This is a simplification, since this does not take the speed of peripherals the processor communicates with (for instance the RAM) into account. In this thesis it is assumed that the clock frequency is decreased with respect to the maximum speed, which has as result that the peripheral devices have to wait for the processor. Then the processor will operate at the lower clock frequency, but spends relatively less time waiting for other devices. This means that assuming the speed scales linearly with the clock frequency is a pessimistic assumption.

For this reason, the work that is done by a task can be expressed as a number of clock cycles. It is sometimes desirable to consider the work as a continuous variable instead of a discrete variable. The number of clock cycles is often a relatively big number and can vary between different executions of some task. Therefore, it is reasonable to consider the number of clock cycles as a continuous variable in the model.

The voltage and frequency pair at which a processor can operate is called an *operating point*. For each operating point, the power consumption can be given. Since the actual voltage is not used in this thesis, it is often not given for brevity.

The power consumption of a computer processor consists of dynamic power and static (or leakage) power. The dynamic power depends on the clock frequency, while the static power is constant and is independent of the clock frequency.

A popular model for the dynamic power of a processor is

$$P_D(f) = ACV^2 f,$$

where P_D is the dynamic power in Watts, A is the switching activity (number of switches between digital 0 and 1), C is the switched capacitance, V is the voltage and f is the clock frequency in clock cycles per second. In order to understand this thesis, it is sufficient to know that both A and C depend on both the processor and the application that is executed on a processor. It will be assumed that A and C are constant for a given processor and application. The voltage V has to be increased if the clock frequency f is increased. It will be assumed that the voltage is scaled linearly with the clock frequency, this model is a.o. used in [9, 12, 18]. The dynamic power can now be written as

$$P_D(f) = \beta f^3,$$

for some given positive constant β. Some authors even use

$$P_D(f) = \beta f^q.$$

for some constant $q \geq 1$ (e.g. [18]). In this thesis the speed of the processor is given by a scaling factor. This is a number between zero and one, where zero is a full stop and one stands for full speed. The scaling factor is obtained by dividing the clock frequency by the highest possible clock frequency.

The static power is a constant P_s and depends on physical characteristics of the processor technology. The total power can now be written as a function of f:

$$P(f) = \beta f^q + P_\mathrm{s}.$$

The models for power consumption change when the processor technology changes. Many processors only allow a finite number of different clock frequencies. An example of such a

Table 1.1: Power dissipation at certain operating points (PowerPC 405LP)

Scaling factor (α)	**Clock Freq. (MHz)**	**Power (Watts) (p)**
0.1	33	0.019
0.3	100	0.072
0.8	266	0.6
1	333	0.75

processor is the PowerPC 405LP processor [7], as given in table 1.1. Note that this processor does not correspond to a model where $q = 3$ and $P_\mathrm{s} = 0$, as used in many articles (e.g., [18, 9]).

Although there are many clock frequencies to choose from, there are costs involved with switching between clock frequencies in terms of energy and time. For the applications in this thesis, it is assumed that the clock frequency is changed only rarely. In case of a video decoder, the clock frequency will be switched every 40ms at most (i.e., for every frame).

The authors of [12] mention that it can take as much as 50μs to 100μs to switch to a different clock frequency. It is expected to decrease further for newer processors. Although some energy and time will be spent on switching the clock frequency, it is very often assumed that these costs can be neglected in comparison to the gain to be expected. Also in this thesis, it is assumed that the costs are negligible.

1.3 Video decoding

A video decoder is an application that is relatively easy to understand, yet still complex enough to be interesting. Therefore it is used as an example throughout this thesis.

Video

To understand video decoding, one has to be familiar with video. In this thesis, when video is mentioned, DVD PAL video is meant. Other digital video formats are similar to what is described in this section. A video sequence is a sequence of so-called video frames, still images that are shown fast enough to let the viewer perceive movement. The number of frames in a second is called the *frame rate*, for DVD PAL this is 25 frames/second. The time between two frames is 40ms. If a frame is ready at a time later than 40ms after the previous frame was shown, it will not be displayed. This frame is then discarded. This is called a *frame drop*. Frame drops should be avoided, since they degrade the quality of the displayed video. Each frame consists of 576 lines, while each line consists of 720 pixels. A pixel is defined using a red, green and blue intensity. This intensity is an integer value in the interval $[0, 255]$, this is represented using an unsigned byte. Hence, $255^3 = 16,581,375$ different colours can be shown by each pixel and three bytes are used to store a single pixel.

Note that if video would be stored without compression, a single second of video would require $3 \times 720 \times 576 \times 25 = 31,104,000$ bytes, which is almost 30MB. Since a DVD has a storage capacity of approximately 4.4GB, this is only enough for approximately 150 seconds of uncompressed video.

Video compression

Clearly, some techniques are required to compress the video data such that they can be stored on a single DVD. There are different types of compression. Although popular compression tools like *zip* can be used for compression, the compression factor of these tools is not sufficient to reduce the size such that the video sequence can be stored on a DVD. Instead intraframe and interframe video compression are used. With intraframe compression, all frames are individually compressed using image compression techniques, which is similar to JPEG compression. Besides using image compression techniques, temporal correlation between frames is used. Consecutive video frames are often very similar, hence this similarity can be exploited by video compression software. Video compression is called lossy when during video compression some information is discarded and cannot be recovered during decompression. The video compression techniques that are not lossy are called lossless. When decompressing an interframe compressed video sequence, it is not possible to decode single video frames, only certain sequences of frames can be decompressed. Lossless and intraframe video compression is often used in television studios, where video sequences are edited and high quality standards are used. For consumer products, lossy and interframe video is desired as the compression ratios are high, which brings down costs significantly. Video compression is often called *video encoding*. The (semi-)reverse process of decompression is often referred to as *video decoding*. The application that is capable of decoding video is called a *video decoder*.

Image compression

If one would consider all frames in a video sequence as independent images, it would be possible to use image compression techniques to reduce the size of the video sequence. The human visual system, which consists of the eyes and a part of the brain, is sensitive to low frequencies and contrast. On the other hand, if high frequencies and colour information are partially discarded, the human visual system will barely notice this. By discarding 75% of all colour information, the number of bytes required for a video frame is halved. Even more can be gained by discarding high frequency information and by using techniques from information theory. In information theory, a random sequence is considered to contain a lot of information if it has a low probability of occurring, while it has little information when the probability is high. These probabilities can again be used for data compression. In practice Huffman coding and arithmetic coding are often used for this. This form of image compression is a form of lossy compression, as the original image cannot be reconstructed, only approximated. However the human visual system does not notice much of the differences between the approximation and the original. These techniques are used in JPEG, but can also be found in intraframe video compression techniques like Motion JPEG (MJPEG). Also for interframe video compression like MPEG-2, MPEG-4 and H-264, lossy image compression is used.

Motion compensation

In many video sequences, the frames only vary a little over time. For instance, an object moves over the screen, the camera itself moves or the lighting changes.

Consider a (current) frame j which is very similar (i.e., a high temporal correlation) to a frame i which was decoded in the past. This frame is used as a reference. A part of the video encoder, called the motion estimator, is used to find a way to reconstruct frame j using the reference frame i. First, the frame i is subdivided into so-called *macroblocks* of 8×8 or

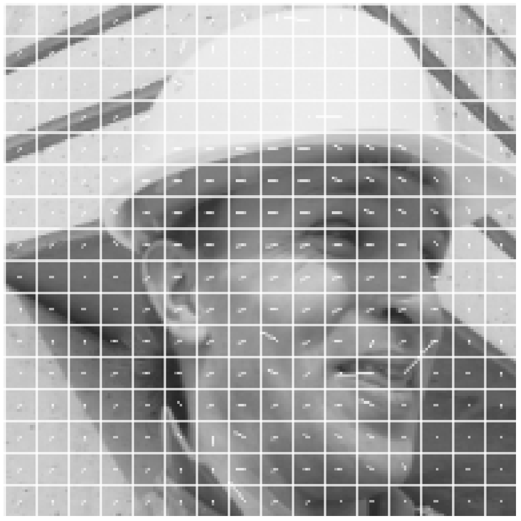


Figure 1.1: Reference frame with macroblocks and motion vectors (from [15])

16×16 pixels. Each macroblock has at most one motion vector, which is used to determine the position of this macroblock in a new frame j'. Frame j' can be constructed by moving the macroblocks from frame i along their motion vectors. The motion estimator tries to find motion vectors, such that the difference between frame h and frame h' will be minimal. The difference between frame j and frame j' is called the residue. The residue is stored using image compression, this can be very efficient since in an ideal case the residue contains almost no information, while the motion vectors are stored separately.

For video decoding, almost the reverse process takes place. The motion compensation, part of the video decoder, is used to construct frame j' using frame i and the motion vectors. After adding the residue to frame j', a very good approximation of frame j is reconstructed. This reconstruction of frame j will be displayed.

An example is given in figure 1.1, where a reference frame is shown together with the motion vector of each macro block. It is shown here that the background stands still, while the head of the foreman moves.

In figure 1.2, a residue of a different frame from the same video sequence is shown. From this residue it is clear that not the entire frame can be reconstructed without this residue. It can also be seen that the residue contains little information, hence image compression will be very efficient.

In many video encoders and decoders the motion estimation and compensation process is very advanced in order to allow for high compression ratios. Most modern video encoders support multiple reference frames, for instance. Techniques like multiple reference frames, subpixel motion compensation, overlapping blocks, motion vector prediction, global motion compensation, etc. make motion compensation very complex and computationally intensive.

Computational effort

For motion compensation, the main task is copying macroblocks (by means of addition) to a video frame. Also some additional processing, for instance interpolation, is required.

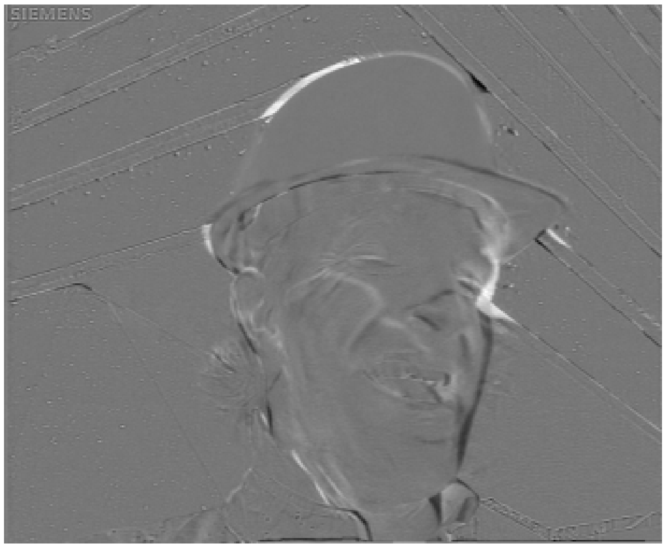


Figure 1.2: Residue (from [3])

The amount of interpolation, the number of reference frames, etc. will be different for each frame. This influences the clock cycles required for processing. The direction of a motion vector influences the order in which the memory of the computer is accessed, which has a direct influence on how efficient the memory cache in a processor is used. This cache has an enormous influence on the performance of the video decoder. In order to predict the processing associated with a video frame, one has to know many details of the computer (processor, memory, etc.), the direction of the motion vectors, the compression techniques used for each frame, etc.

Consider a black video frame. No previous frames are required to reconstruct the current frame, hence little processing is required. The other extreme is a video sequence with lot of movement, where motion vectors point in all directions. In that case, the cache is not used efficiently and it will take relatively long to decode this frame.

The temporal correlation between frames can again be used here. The motion vectors and complexity of consecutive frames can be expected to be similar. To predict the computational effort for motion compensation of frame i, taking computational effort of the previous frame of the same type is often a reasonably good predictor. One can also use other properties of the video frame for prediction.

The video decoder does much more than just motion compensation, hence it is very difficult to predict the work associated with a frame. Prediction of work is not a subject of this thesis, although this is a popular research topic. For this thesis it is important to know that the execution time for decoding a frame is variable and often significantly below the maximum execution time.

In figure 1.3, a plot of execution times of a motion compensator for HD video is shown for a part of the sequence. It can be seen that the execution times appear to be correlated. The minimum execution time of the entire sequence is 5.36ms, the average execution time is 8.09ms, while the maximum execution time is 11.90ms.

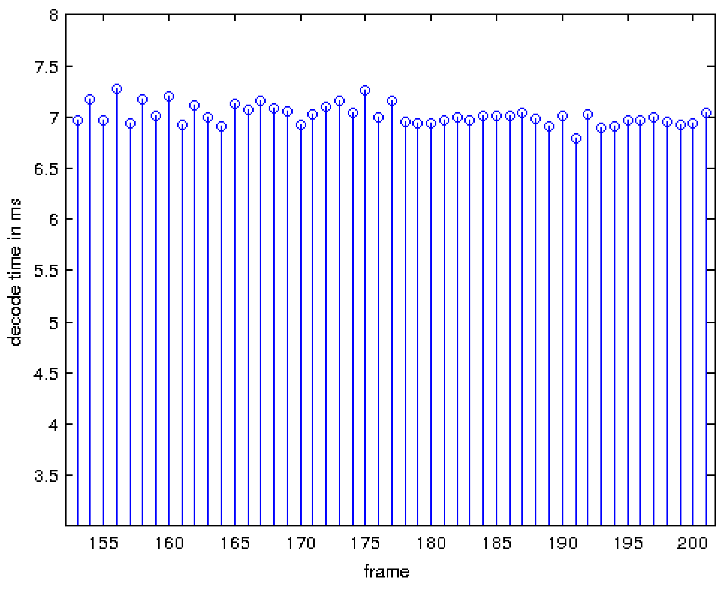


Figure 1.3: Execution times for motion compensation

1.4 Terminology and definitions

In the previous sections, an informal introduction of the energy minimisation problem was given. In section 1.2, a model of a processor was discussed, where the work is given as the number clock cycles and power in Watts. In the remainder of this thesis, work and power are considered abstractly and unitless. The same model can then, for instance, also be applied to hard disk drives and communication systems, where respectively blocks and bits can be used to indicate work. As long as the work is relatively long compared to the switching time and costs, energy consumption can be reduced.

The speed of the device can be scaled using a scaling factor.

Definition 1.1 (Set of scaling factors) The scaling factor is a value from the set A, given by $A = [L, 1]$, with $L \in \mathbb{R}^+$. The value L is the greatest lower bound of the scaling factor. □

Central are the definitions of a task and a process:

Definition 1.2 (Task) A task is a quadruple (w, t, d, α) where

- $w \in \mathbb{R}^+$, referred to as the *work*
- $t \in \mathbb{R}^+$, referred to as the *execution time*
- $d \in \mathbb{R}^+$ is the *deadline*
- $\alpha : [0, t] \to A$, referred to as the *scaling function* □

A task describes work that has to be finished before a certain deadline. Vice versa, any quantity of work that has a deadline and of which the speed can be scaled, can be modelled as a task. For instance, in the context of a video decoder, decoding a single frame can be considered a task. The scaling function assigns a *scaling factor*, a value in the set A, to a time instance relative to the beginning of the task. To describe multiple tasks, the notion of a *process* is introduced.

Definition 1.3 (Process) A process is a triple $((\mathbb{T}_n), p, w^{(0)})$ where

- $(\mathbb{T}_n)$ is a sequence $((w_1, t_1, d_1, \alpha_1), (w_2, t_2, d_2, \alpha_2), \dots)$ of tasks, where $d_1 < d_2 < \dots$. This is either a finite sequence (of length N) or an infinite sequence.
- $p : A \to [0, \infty)$, referred to as the *power function*, is a convex function, with $\frac{p(\alpha)}{\alpha}$ non-decreasing, twice differentiable and convex.
- $w^{(0)} \in \mathbb{R}^+$, referred to as the work per time at the highest speed. □

In this chapter the sequence $(\mathbb{T}_n)$ is assumed to be finite. For brevity, the sequences (w_n), (t_n), (d_n) and (α_n) indicate the sequences of work, execution times, deadlines and scaling functions associated with the sequence of tasks $(\mathbb{T}_n)$ respectively. In this thesis, the relation $\bar{p}(\alpha) := \frac{p(\alpha)}{\alpha}$ is used very often since it expresses the energy per work. This is an important quantity as it does not depend on time. As the costs in optimisation problems in this thesis depend on $p(\alpha)$ and/or $\bar{p}(\alpha)$, the properties for these functions are required for both technical and modelling reasons.

The total amount of work is related to the execution time. For one task, the work done in a unit of time and the scaling function are related by

$$w = \int_0^t w^{(0)} \alpha(\tau) d\tau. \tag{1.1}$$

Here $w^{(0)}\alpha(\tau)$ is the work per time at the speed $\alpha(\tau)$. Integrating this over time gives work.

Associated with a process is the sequence of start times (b_n), defined as

Definition 1.4 (start times (b_n))
$$b_n := \sum_{i=1}^{n-1} t_i$$

and the sequence of completion times (f_n) is defined as

Definition 1.5 (completion times (f_n))
$$f_n := b_n + t_n = \sum_{i=1}^{n} t_i.$$

Example 1.1 (Video decoder) Consider the video decoder process which has to decode a sequence of video frames. Decoding a single frame is a task. When there are N frames, there are N tasks. The number of clock cycles that are required to decode a single frame i is given by w_i. In figure 1.3, $\frac{w_i}{w^{(0)}}$ is shown for the motion compensation part of a video decoder. From this figure it is clear that w_i is not the same for every $i \in \{1, \ldots, N\}$. With a frame rate of 25 frames/second, every 40ms a frame has to be displayed. The deadlines are then $d_1 = 40$, $d_2 = 80$, $d_3 = 120$, etc.

To ease the notation in many definitions and calculations, $d_0 := 0$ is defined.

In practice, many processes have the following property.

Definition 1.6 (Admissible process) A process is admissible if for all $n \in \{1, \ldots, N\}$

$$d_{n-1} + \frac{w_n}{w^{(0)}} \leq d_n. \qquad \square$$

This implies that if $\alpha_i = 1$ for all i, the completion time is before the deadline. This means that at the highest speed, the deadline of task n will be met when the task begins at the deadline of the previous task. Throughout this thesis, it is assumed that all processes are admissible.

If a device allows only for a finite number of scaling factors, some additional definitions are required. First the definition of a scaling factor is redefined:

Definition 1.7 (Set of scaling factors (finite set)) The finite set A is the set of scaling factors, given by $A = \{\bar{\alpha}_1, \ldots, \bar{\alpha}_M\} \subset (0, 1]$ with $\alpha_M = 1$ and $\alpha_1 < \cdots < \alpha_M$. Here $\alpha_1, \ldots, \alpha_M$ are the scaling factors. The value M is the number of scaling factors in A, i.e. $M = |A|$. $\square$

Assume A is a finite set. Now the function $\alpha(\tau)$ can only assume the M different values in A. If a task n is executed and commits w_n work, this work is distributed over M scaling factors. For a fraction of the work, namely $r_{n,1}$, the scaling factor α_1 is used. For a different fraction of the work, namely $r_{n,2}$, the scaling factor α_2 is used, etc. The exact definition of the fraction of work $r_{n,m}$ is given in the following definition.

Definition 1.8 (Fraction of work $r_{n,m}$) For each task $n \in \{1, \ldots, N\}$ and each scaling factor $m \in \{1, \ldots, M\}$, $r_{n,m}$ is the amount of work of task n for which the scaling factor m is used. For a given task n, the sum of the fractions of work for all scaling factors is the work, i.e.

$$\sum_{m=1}^{M} r_{n,m} = w_n. \qquad \square$$

The time between the deadline of task i and task $i+1$ is given by

$$\Delta d_i := d_{i+1} - d_i.$$

For many processes, the time between the deadlines of tasks increases with a constant. This is is special case of periodic process.

Definition 1.9 (Periodic process) A process is called periodic with period T if there is a value $T \in \mathbb{N}$ such that for all $n \in \{1, \ldots, N\}$ and $n + T \leq N$:

$$\Delta d_n = \Delta d_{n+T}$$

If the process is periodic with period 1, there is a value $D \in \mathbb{N}$, such that:

$$(d_n) = (D, 2D, \ldots, ND)$$ □

The video decoder process is an example of a periodic process. The time between deadlines is 40ms, hence the process is periodic with period 1. Another example of a periodic process is the following:

Example 1.2 Consider $(w_1, w_2, \ldots, w_9) = (10, 5, 7, 9, 8, 1, 7, 9, 10)$, $D = 20$, $(d_1, d_2, \ldots, d_9) = (D, 2D, \ldots, 9D)$ and $w^{(0)} = 1$.

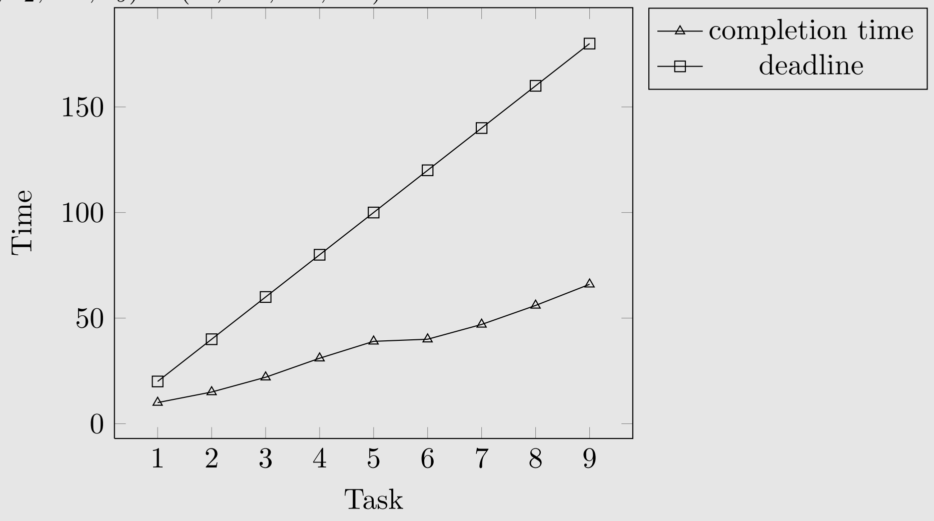


Figure 1.4: Example process

Assume the scaling functions are given by $\alpha_i(\tau) = 1$ for all $i \in \{1, \ldots, N\}$, this corresponds to running at full speed. Figure 1.4 shows the completion times $((f_n))$ and the deadlines $((d_n))$. The tasks finish well before their deadlines. This can be seen from the figure, since the graph that shows the completion times is below the graph that shows the deadlines.

An alternative method of showing this process is given in figure 1.5. In this figure, the time is plotted against the cumulative work. The cumulative work of task n is the sum of the work of the first n tasks.

In figure 1.5, time is plotted against the cumulative work. This type of figures are used in the literature (for instance [8]) since they can show the work, deadlines, execution times and scaling factors. The graph corresponding to running the tasks at the maximum speed shows the cumulative work that has been done at a certain time instant. The graph indicating the deadlines are shown in the figure. This graph is below the graph that shows the tasks running at maximum speed. This indicates that at deadlines, more work is done than enforced by the deadlines.

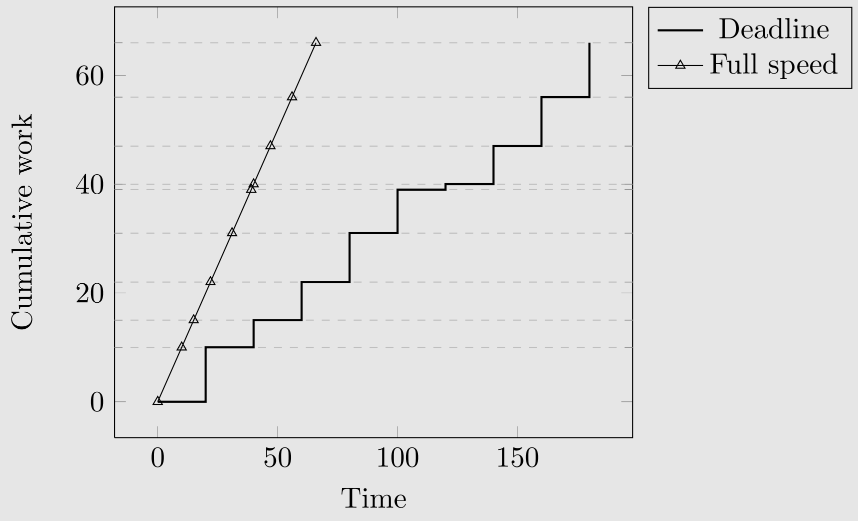


Figure 1.5: Example process

1.5 Dynamic Voltage and Frequency Scaling

Dynamic Voltage and Frequency Scaling (DVFS) gives the software control over the voltage and frequency of a processor. It was already mentioned that voltage and frequency influence the energy consumption of the processor. DVFS can be used to decrease energy consumption, without decreasing the quality.

Consider the processor shown in table 1.1. At full speed 0.1473W is used, while at halve of the speed (scaling factor 0.5) only 0.0380W is used. The processor will run twice as long, but consume less power. If a task i takes t_i seconds to run at full speed, it will take $2t_i$ seconds at halve of the speed. The energy consumption will decrease from $0.1473t_iJ$ to $2t_i \times 0.0380J = 0.076t_iJ$. Energy consumption is decreased this way, but it is important to check if no deadline is missed, since the execution time is now doubled.

Some processors are capable of DVFS, but not all of them are useful for our purposes. There are operating points that are in practice inefficient, these operating points are called *power inefficient*. Consider the PowerPC 405GP processor, given in [13]. It is has the operating points as given in table 1.2.

In case of the PowerPC 405GP processor, running for a time t_i seconds will consume $3.13t_iJ$. For a scaling factor of 0.5, the same task will require $5.26t_iJ$. This shows that for the PowerPC 405GP processor, running at a lower scaling factor will require more energy for the entire task. Hence, if p is increasing, it can still be the case that choosing a lower scaling factor will consume more energy. For this reason, instead of using power (energy/second) given by p, it is better to consider the energy/work as given by $\bar{p}$. Now table 1.1 is extended with energy/work as given in table 1.3. Since the energy/work ($\bar{p}$) is required to be an increasing function of the scaling factor, energy consumption can be reduced by decreasing the speed. This requirement is not true for the PowerPC 405GP processor.

If energy/work is considered, PowerPC 405LP processor is clearly a favourable processor

Table 1.2: Power dissipation at certain operating points (PowerPC 405GP)

Scaling factor (α)	**Clock Freq. (MHz)**	**Power (Watts) (p)**	**Energy/work ($\bar{p}$)**
0.248	66	2.27	9.153
0.5	133	2.63	5.26
0.752	200	2.89	3.843
1	266	3.13	3.13

Table 1.3: Power dissipation at certain operating points (PowerPC 405LP)

Scaling factor (α)	**Clock Freq. (MHz)**	**Power (Watts) (p)**	**Energy/work ($\bar{p}$)**
0.1	33	0.019	0.19
0.3	100	0.072	0.24
0.8	266	0.6	0.75
1	333	0.75	0.75

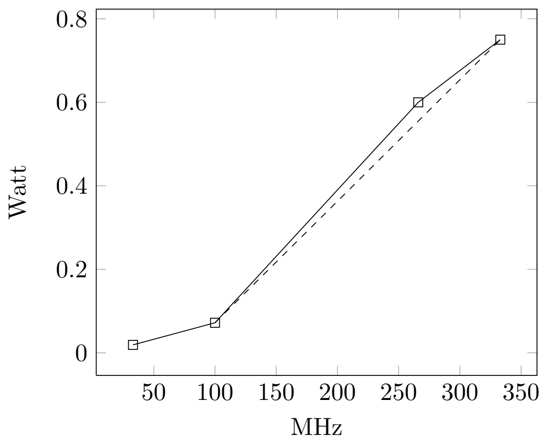
while the PowerPC 405GP is not a desirable processor in this regard. Note that if the application is not real-time, the PowerPC 405GP can still be useful. If this processor would be used in a laptop, the user can increase the speed of the computer to save energy.

When using the PowerPC 405LP as shown in table 1.3, one should be careful. As mentioned in [7], the clock frequency 266 MHz is power inefficient. The authors observe that the operating point at 266MHz can be emulated by running at 100 MHz for a part of the time and at 333MHz for the remainder of the time. This is illustrated in figure 1.6a, where the operating points of the processor are shown at 33, 100, 266 and 333MHz with their respective power consumption. The dashed line indicates the power consumption if one would *emulate* a clock frequency by running at neighbouring clock frequencies such that the average clock frequency is the desired clock frequency. Emulating the clock frequency will result in a lower power consumption than actually running at 266MHz. This is the reason why the function p has to be convex, since that means all inefficient clock frequencies were discarded from the model.

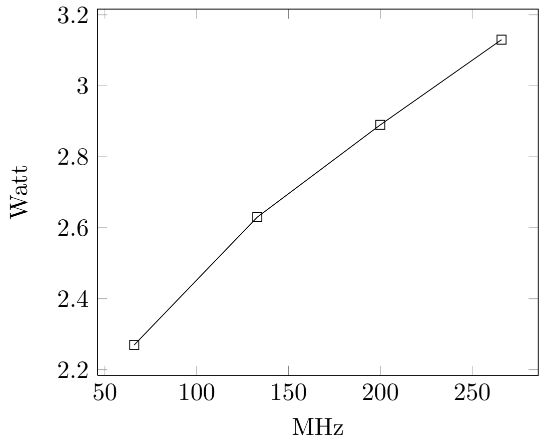
In case all scaling factors in $[L, 1]$ can be used, energy/work is important. Consider, for example, the function $p(\alpha) = \beta\alpha^3 + \gamma$. The function $\bar{p}(\alpha) = \beta\alpha^2 + \frac{\gamma}{\alpha}$ has as derivative $\bar{p}'(\alpha) = 2\beta\alpha - \frac{\gamma}{\alpha^2}$ and is non-decreasing if and only if $\alpha \geq \sqrt[3]{\frac{\gamma}{2\beta}}$. This is demonstrated for $\bar{p} = 0.2\alpha^2 + \frac{0.01}{\alpha}$ in figure 1.7

Now take $L = \sqrt[3]{\frac{\gamma}{2\beta}}$. It is beneficial to use L as a lower bound for the scaling factor; below L the function $\bar{p}$ is decreasing. The costs associated with scaling factor L are lower than the costs associated with scaling factors below L, while the task is only finished earlier. For processors with a power function as is given here, DVFS can no longer be used efficiently when $\frac{\gamma}{2\beta} \geq 1$, because the function $\bar{p}$ is decreasing on the interval $[0, 1]$. In several articles, it is implicitly assumed that $L = 0$ (e.g., [7, 9, 18]). The techniques in these articles can then only be applied efficiently when $\bar{p}$ is increasing on the interval $[0, 1]$. Since in practice $\gamma > 0$, the techniques for DVFS need to take L into account.

When DVFS cannot be used efficiently (i.e., $L = 1$), it is best to operate using the scaling



(a) PowerPC 405LP



(b) PowerPC 405GP

Figure 1.6: Power for a given clock frequency

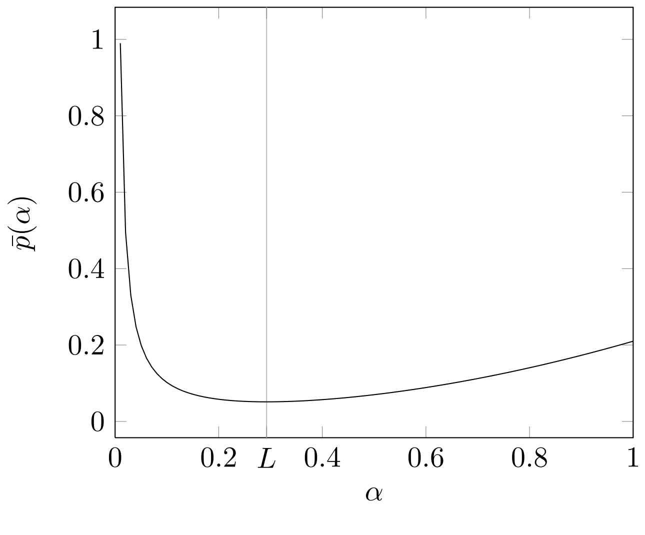


Figure 1.7: Plot of $\bar{p} = 0.2\alpha^2 + \frac{0.01}{\alpha}$

factor 1. It is assumed throughout this thesis that no energy is used after the last task is finished. In that case a different application can be started or the processor can be powered off.

1.6 Offline optimisation of scaling factors

Applications of offline optimisation

Firstly, it will be assumed that for all $i \in \{1, \ldots, N\}$, the work w_i is known. Using the given work (w_n) and the given deadlines (d_n), the scaling factors are calculated such that the energy consumption is minimal. This is important for two reasons. The first reason is that offline optimisation is a first step towards solving the harder online optimisation problem, this is the topic of the next section. The second reason is that for some applications offline optimisation is important, as sometimes all work (w_n) and deadlines (d_n) are known before the application is started. Several of the applications for which offline optimisation can be applied will be discussed next.

In [8] a streaming video service is described where video is stored on a video streaming server which knows the work associated with each video frame. The scaling factor, together with the actual video frame is sent to each client computer. The client computer can then decode the video frames at the optimal speed, such that energy is minimised.

Another example is a digital video recorder, which can store the work associated with a video frame while recording the video. This is possible because as part of the encoding process, the video is decoded.

Portable game devices like the Sony Playstation Portable (PSP) and the Nintendo 3DS (3DS) are capable of playing back video. On the PSP, the video is distributed on a UMD disk or using digital distribution over the Internet. On the 3DS it is possible to watch 3D videos, which requires decoding two images (one for each eye) for each video frame. The battery of the 3DS has just enough charge to operate for only three to five hours when using the 3D features, showing the energy consumption increases when new features like 3D playback are added, while battery technology cannot keep up with this development. The video files for the 3DS are distributed over the Internet. It would be technologically possible for the manufacturer to distribute the values for w_i or even the optimal functions α_i together with the actual video content. This will decrease the power consumption for video playback. For users this means that the device can be used for a longer time without recharging the battery.

In these applications, it is assumed that before execution, future work is somehow known. The optimisation takes place before the tasks are actually executed. Because of this, this form of optimisation is called offline optimisation. As said, for some applications the information required for offline optimisation can be provided, but this is not always the case or desirable. When this is not the case, heuristics can be used to save energy, a lot of research is devoted to such systems. Offline optimisation can still be useful for such applications. First of all, offline optimisation can be used to evaluate the performance of online heuristics. Second, lessons can be learned from offline optimisation that can be used when designing heuristics.

Trade-offs

For energy efficient operation, the designer of a system is free to choose $\alpha_i(\tau)$ and t_i for each task i. Assume, for the moment, that for all i, $\alpha_i(\tau)$ is a constant function, $w^{(0)} = 1$ and

$p(\alpha) = \alpha^3$. From equation (1.1), it is clear that $t_i = \frac{w_i}{\alpha_i}$, showing for $N = 1$, only α_1 has to be determined and for $N = 2$, only α_1 and α_2 have to be determined. If there are two tasks ($N = 2$), it can already be seen how tasks influence each other. this is illustrated using the following example.

Example 1.3 In this example the energy optimisation problem is informally discussed and it is assumed that the scaling functions are constant functions. Assume $N = 1$, $w_1 = 10$, $d_1 = 20$ and $p(\alpha) = \alpha^3$. Then the energy per work is $\bar{p}(\alpha) = \alpha^2$. Since this function is non-decreasing, the minimum is attained at $\alpha_1(\tau) = \frac{1}{2}$.

Now assume $N = 2$, $w_1 = 10$, $w_2 = 20$, $d_1 = 20$, $d_2 = 40$ and $p(\alpha) = \alpha^3$. If the solution for $N = 1$ is used, $\alpha_1(\tau) = \frac{1}{2}$ and task 2 can begin execution at time 20. However, the only feasible choice for $\alpha_2(\tau)$ is $\alpha_2(\tau) = 1$. This gives the costs

$$\bar{p}(\alpha_1(\tau))w_1 + \bar{p}(\alpha_2(\tau))w_2 = \left(\frac{1}{2}\right)^2 \times 10 + 1^2 \times 20 = 22.5$$

but if $\alpha_1(\tau) = \frac{3}{4}$ and $\alpha_2(\tau) = \frac{3}{4}$ the deadlines will still be met and

$$\bar{p}(\alpha_1(\tau))w_1 + \bar{p}(\alpha_2(\tau))w_2 = \left(\frac{3}{4}\right)^2 \times 10 + \left(\frac{3}{4}\right)^2 \times 20 = 6.875,$$

which shows the costs have decreased. This shows that in finding the minimum, the tasks interact. To determine the optimal solution for task n, not only tasks up to task n should be considered, but also the entire future.

The example illustrates that decreasing the speed to the minimum speed that is allowed does not necessarily give the optimal solution. Finding the optimal solution is a difficult problem and will be discussed in chapter 3.

1.7 Online optimisation of scaling factors

Applications of online optimisation

It is not always the case that all the values w_i can be known before task i is executed. This makes it impossible to use offline optimisation. An example of such application is video broadcasting, where video frames are distributed over the Internet, satellite, a cable television network, etc. to various devices. If this is a live broadcast, it is clear that the future work is not known. The same is true for many communication applications.

Optimisation

Many online algorithms use a heuristic that decreases the energy consumption, but the minimum is not attained. To explain the problems encountered in online optimisation, the following concepts are introduced.

Definition 1.10 (Worst Case Work) The Worst Case Work (WCW), denoted by $W \in \mathbb{R}^+$, is the smallest value for all possible sequences (w_n) such that:

$$w_i \leq W \qquad \text{for all } i \in \{1, \dots, N\}$$

□

In practice the WCW is hard to determine, instead the Measured Worst Case Work (MWCW) is determined. This is the largest amount of work that is encountered when running the application for various inputs. In this thesis, it is assumed that W is known. Often the WCW is significantly higher than the average work. In that case the difference between the deadline and the time instant at which the task finishes becomes relevant. This difference is called the slack time of a task, as given in the following definition.

Definition 1.11 (Slack time) For task n, the slack (or slack time) $s_n \in \mathbb{R}_0^+$ is defined as

$$s_n := d_{n-1} - f_{n-1}.$$

□

This slack time s_n can be used to decrease the scaling factor for future tasks $n+1, \dots, N$. It is required that the deadlines for the tasks $1, \dots, n-1$ are met, hence $f_n \leq d_n$. If $s_n > 0$, the tasks $n, \dots, N$ together get an additional time s_n to execute. It is relevant how this slack time is distributed among all future tasks. For proper minimisation, it is important to have a good prediction of future work. Chapter 4 explains how to exploit slack time such that the energy consumption is minimised in this setting.

CHAPTER 2

Related work

In this chapter related work from the literature is discussed. For a proper discussion, it is often sufficient to look at processes that consist of a single task. The topic of section 2.1 is offline optimisation. In section 2.2, online optimisation is discussed.

2.1 Offline optimisation

Optimisation problem

To discuss some results from the literature, it is assumed for brevity that there is only a single task ($N = 1$). The energy minimisation problem for a single task can be written as:

$$\begin{aligned} \min_{\alpha(\tau),t} \quad & \int_0^t p(\alpha(\tau))d\tau \\ \text{s.t.} \quad & \int_0^t w^{(0)}\alpha(\tau)d\tau = w \\ & t \leq d. \end{aligned}$$

This is an infinite dimensional problem in the non-trivial case ($w \neq d$), since a solution $\alpha(\tau)$ is desired for every $0 \leq \tau \leq t$. For now, it is assumed that if p is convex and if a solution exists, there always exists a solution $\alpha(\tau)$ that is a constant function. In the next chapter, this will be discussed in detail.

In the remainder of this chapter, $L = 0$, $w^{(0)}$ and the scaling factor $\alpha \in (0, 1]$ will be used, instead of the function $\alpha(t)$ because (if there is a solution) a constant solution exists. Now the optimisation problem can be denoted as:

$$\begin{aligned} \min_{\alpha} \quad & \int_0^t p(\alpha)d\tau \\ \text{s.t.} \quad & t\alpha = w \\ & t \leq d. \end{aligned}$$

By substituting $t = \frac{w}{\alpha}$ and calculating the integral (note that $p(\alpha)$ is constant), this problem can be written as:

$$\begin{aligned}\min_{\alpha(\tau)} & \frac{p(\alpha)}{\alpha} w \\ \text{s.t.} & \frac{w}{\alpha} \leq d \\ & \alpha \leq 1\end{aligned}$$

or since $\bar{p}(\alpha) = \frac{p(\alpha)}{\alpha}$:

$$\min_{\alpha} \left\{ \bar{p}(\alpha) w_i \,\middle|\, \frac{w}{d} \leq \alpha \leq 1 \right\}.$$

This means that the minimum of the function $\bar{p}$ has to be found on the interval $[\frac{w}{d}, 1]$. If the function $\bar{p}$ is monotonic, this is trivial. When $\bar{p}$ is strictly decreasing, the optimal solution is $\alpha = 1$. When $\bar{p}$ is strictly increasing, the optimal solution is $\alpha = \frac{w}{d}$.

Solution in the literature

In [18], the problem is generalised for multiple tasks. In this paper, it is claimed that the solution of the problem does not depend on the power function p, it is only required that p is convex. Hence, if one finds an optimal solution, it is an optimal solution for all convex power functions. Unfortunately, this only holds when in addition to p is convex, also $\bar{p}$ is convex and increasing. If $\bar{p}$ is not convex or not increasing, a counter example can be constructed to show that a single solution for this problem can not exist for all p.

First assume $p_1(\alpha) = \alpha^2$, which is a strictly increasing convex function. Then according to the previous section, the minimum of $\frac{\alpha^2}{\alpha} = \alpha$ has to be found. Since this is a strictly increasing function, the optimal solution is $\alpha = \frac{w}{d}$. Now assume $p_2(\alpha) = e^{\alpha}$, which is also a strictly increasing convex function. However, $\bar{p}_2(\alpha) = \frac{e^{\alpha}}{\alpha}$ is a strictly decreasing function on $(0, 1]$, see figure 2.1. Because of this, $\alpha = 1$ is the optimal solution. The minimiser of $\bar{p}_1$ is a maximiser of $\bar{p}_2$ and vice versa. This demonstrates that it is not possible to find a single value α that minimises all functions $\bar{p}(\alpha) = \frac{p(\alpha)}{\alpha}$ when p is convex. However, if the function $\bar{p}$ is increasing and convex, it is possible.

In [8], a generalisation of the theory in [18] is discussed. In this article buffers are considered and constraints are added to ensure the buffers do not overflow. The data in the buffers are modelled at byte level accuracy, which makes the algorithm in this article very general and widely applicable. Although the paper contains a proof for p is convex, the algorithm does not depend on p itself. For this result to hold, also the constraints on $\bar{p}$ have to be considered, otherwise a counter example can again be found.

Finite number of scaling factors

It was assumed that $\alpha(t)$ can attain the values in the range $[0, 1]$. However, many processors offer only a fixed number of *operating points* one can choose from. The set of scaling factors one can choose from is denoted by $A \subset (0, 1]$. In the remainder of this section, it is assumed A contains only two values, namely $\bar{\alpha}_1$ and $\bar{\alpha}_2$ ($A = \{\bar{\alpha}_1, \bar{\alpha}_2\}$) with $\bar{\alpha}_1 < \bar{\alpha}_2$. Now the optimisation problem

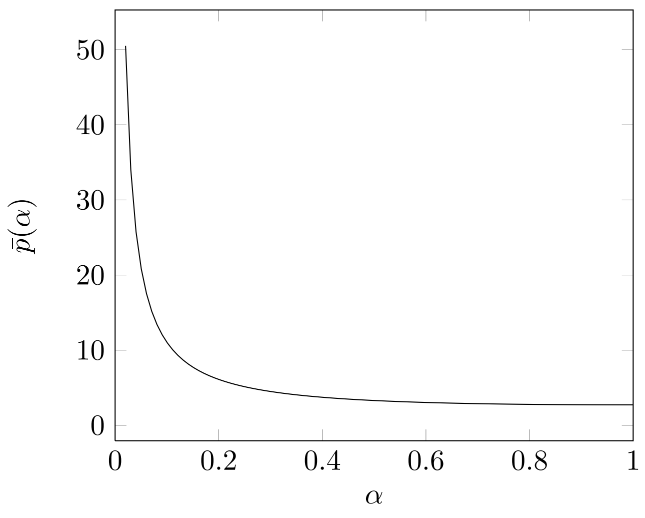


Figure 2.1: Plot of $\bar{p}(\alpha) = \frac{e^\alpha}{\alpha}$

$$\begin{aligned} \min_{\alpha(\tau),t} \quad & \int_0^t p(\alpha(\tau))d\tau \\ \text{s.t. } & t\alpha = w \\ & t \leq d \end{aligned}$$

can be rewritten to

$$\begin{aligned} \min_{r_1,r_2} \ & \bar{p}(\bar{\alpha}_1)r_1 + \bar{p}(\bar{\alpha}_2)r_2 \\ \text{s.t.} & \frac{r_1}{\bar{\alpha}_1} + \frac{r_2}{\bar{\alpha}_2} = w \\ & r_1 + r_2 = w \\ & t \leq d \\ & r_1 \geq 0 \\ & r_2 \geq 0 \end{aligned} \tag{2.1}$$

which is a linear program. Here r_1 and r_2 denote the amount of work done at the speeds given by α_1 and α_2 respectively. Since the total amount of work is w, $r_1 + r_2 = w$. This linear program does not necessarily have a unique optimal solution. It is claimed in [7, 9] that if p is convex, the optimal solution can be found by solving the non-linear optimisation problem $\min_\alpha \{\bar{p}(\alpha)w | \frac{w}{d} \leq \alpha \leq 1\}$. This is done by defining

$$\tilde{\alpha} := \frac{w}{\frac{r_1}{\alpha_1} + \frac{r_2}{\alpha_2}}.$$

Substituting α by $\tilde{\alpha}$ in problem (2.2), as defined below, again gives problem (2.1).

$$\begin{aligned}
&\min_{r_1,r_2} \left[\bar{p}(\alpha_1)\frac{\tilde{\alpha}-\alpha_2}{\alpha_1-\alpha_2}+\bar{p}(\alpha_2)\frac{\tilde{\alpha}-\alpha_1}{\alpha_2-\alpha_1}\right]\frac{w}{\tilde{\alpha}} \\
&\text{s.t.} \frac{w}{\tilde{\alpha}} \leq d \\
&\quad \tilde{\alpha} \geq \alpha_1 \\
&\quad \tilde{\alpha} \leq \alpha_2.
\end{aligned} \tag{2.2}$$

Now $\tilde{p}(\tilde{\alpha}) = \bar{p}(\alpha_1)\frac{\tilde{\alpha}-\alpha_2}{\alpha_1-\alpha_2}+\bar{p}(\alpha_2)\frac{\tilde{\alpha}-\alpha_1}{\alpha_2-\alpha_1}$ is convex (affine, increasing and thus also convex). Note that this affine function is a line through $(\alpha_1, \bar{p}(\alpha_1))$ and $(\alpha_2, \bar{p}(\alpha_2))$, hence all clock frequencies between α_1 and α_2 can be *emulated.* By the claims in [9], the optimal solution can be found by using the algorithm in [18] since $\tilde{p}$ is convex. Unfortunately $\frac{\tilde{p}(\alpha)}{\alpha}$ is not always convex and non-decreasing. It can be shown that depending on $\bar{p}$, the result might still hold. In [9], only $p(\alpha) = \alpha^3$ is considered and although the proof uses the convexity of p and the claim in [18], the result still holds. However, in [7], it is assumed arbitrary convex functions $\bar{p}$ can be used. It can be shown that not every affine function $\tilde{p}(\tilde{\alpha}) = a + \tilde{\alpha}b$ is non-decreasing, after dividing by $\tilde{\alpha}$, as is done in the optimisation problem above. If a is negative, $\frac{a+\tilde{\alpha}b}{\tilde{\alpha}}$ is a decreasing function. As in section 2.1, a counter example can be constructed, showing the result in [7] holds for p convex and $\bar{p}$ convex and non-decreasing. This problem is related to the lower bound of α as was discussed in section 1.5. Note that this lower bound can become 1 for decreasing power functions, removing all freedom and making the problem static.

2.2 Online optimisation

For online optimisation, several approaches are popular. First the different approaches are discussed, concluding with an evaluation of these approaches.

Predictions

If the application is known, it is possible to predict the work just before execution. In [14], a video decoder is studied. To decode a single frame, n so-called macro blocks have to be decoded. First $\frac{n}{2}$ macroblocks are decoded. The time to decode the first $\frac{n}{2}$ macroblocks is used to predict the time to decode the remaining $\frac{n}{2}$ macroblocks. Using the predicted time and the actual scaling factor, the predicted work can be calculated. In many articles [1, 10, 19], statistics are used to predict work. This is done by either finding a pdf that works well, or by using time series.

In all these approaches, the work is predicted. To preserve as much energy as possible one needs a good strategy to choose good scaling factors.

Strategy

To make sure no deadline is missed, it must be assumed that the work for a task is the worst case work W. By doing this, many tasks will finish early and slack time will start to accumulate. This slack time can be used by future tasks. A good strategy is required to distribute available slack among future tasks.

Many strategies are greedy and choose the lowest scaling factor, i.e. all available slack is used by the first task. This is a greedy approach to DVFS. Some articles even assume

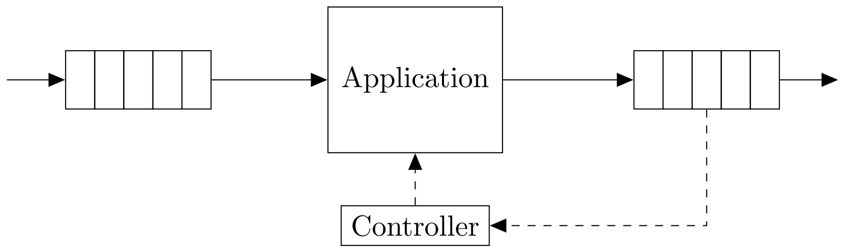


Figure 2.2: Feedback control for DVFS

the prediction is very reliable and decrease the scaling factor to a point where meeting the deadline can not be guaranteed, in case the prediction appears to be very wrong. Examples of work where slack and possibly the prediction is used in a greedy fashion can be found in [10, 14].

The state of the art is given in [19]. The strategy for slack distribution given in this article depends on the prediction of future work. When task $n-1$ is just finished, the optimal scaling factor for task n has to be calculated.

In the optimisation problem given in [19], the size of the feasible region is decreased. Because of this, the problem is not always feasible, while with the original feasible region, the problem is feasible. For the cases the problem becomes infeasible, a heuristic is used to find a solution that gives relatively low costs. With this approach, it is guaranteed that no deadlines are missed. In this thesis, the problem is written such that an optimal solution can always be found.

Feedback control

In several DVFS related articles [12, 17, 11], a feedback control based approach is used. A commonly used structure is shown in figure 2.2.

There are two buffers in this figure. On the left side of the figure, a buffer is shown in which data is placed that has to be processed. This data can, for instance, be a sequence of compressed video frames. On the right side of the figure a buffer is shown that stores output produced by the application. The application consists of tasks that process input from the input buffer and place the results in the output buffer. At every deadline, at least one result should be present in the output buffer. The buffer occupancy level of the output buffer is used to determine the scaling factor of the processor. When the buffer is almost empty, the processor should run at a high speed, while when the buffer is almost full, the processor should run at a low speed. The controller takes care of this and determines the scaling factor based on some given algorithm. This model is used as a basis in many articles. For instance [12] uses a PI regulator to determine the scaling factor, while a non-linear controller for multiple processors is given in [2]. A problem with this approach is that it is unknown whether the system is stable, stays properly inside the given bounds, etc. To deal with this, most controllers have predefined behaviour below a certain minimum level (use a high scaling factor) and above a certain maximum level (use a low scaling factor). Furthermore, it is possible that oscillations occur. No formal properties of these systems are given in these articles and it is not known how close feedback control based systems will approach the optimum. An analytic approach can be found in [17], here the queues of the system are modelled and a non-linear controller is derived.

One might observe that the feedback control approach is a mix of both prediction and strategy. Because both aspects of energy minimisation are mixed, it becomes hard to explain how well this approach actually works.

Chapter 3

Offline energy optimisation

In this chapter the offline optimisation problem is discussed. This problem was introduced in section 1.6. The terminology and definitions that are used in this chapter were introduced in section 1.4. In this chapter, a mathematical model of a process is given that is used to minimise energy. Theory from convex optimisation is used to accomplish this. It is assumed that the reader is familiar with convex optimisation and especially with the Karush Kuhn Tucker (KKT) conditions. An overview of convex optimisation can be found in appendix A.

3.1 Infinite-dimensional problem

Problem description

A process is a sequence of tasks which are executed by, for instance, a device of which the speed can be scaled. The speed at which the work is carried out can be scaled between 0 ("full stop") and 1 ("full speed"). Increasing the speed will increase the costs, while decreasing the speed can result in a missed deadline. The total costs for all tasks can be expressed as

$$\sum_{i=1}^{N} \int_{0}^{t_i} p(\alpha_i(\tau))d\tau$$

where the power is integrated over the time duration of an individual task and the energy for all processes is summed. Meeting all deadlines can be expressed as

$$f_n \leq d_n \qquad \text{for all } n \in \{1, \dots, N\},$$

i.e. the completion time of task n is before the deadline of task n.

The main goal of this chapter is to show how the costs associated with a process can be minimised by choosing proper scaling functions (α_n) and execution times (t_n) such that all work is done and the deadlines will be met. This problem can be written as the infinite-dimensional optimisation problem

$$\begin{aligned}
\min_{\alpha_1(\tau),\ldots,\alpha_N(\tau),t_1,\ldots,t_N} \quad & \sum_{i=1}^{N} \int_0^{t_i} p(\alpha_i(\tau))d\tau \\
\text{s.t.} \quad & \int_0^{t_n} w^{(0)}\alpha_n(\tau)d\tau = w_n \qquad \text{for all } n \in \{1,\ldots,N\}, \qquad (3.1)\\
& \sum_{i=1}^{n} t_i \le d_n \qquad \text{for all } n \in \{1,\ldots,N\}.
\end{aligned}$$

Note that for all admissible processes, a feasible point exists with $\alpha_i = 1$ and $t_i = \frac{w_i}{w^{(0)}}$. Hence, the feasible set is nonempty.

Rewriting to a finite-dimensional problem

To solve optimisation problem (3.1), functions (α_n) have to be found. This makes this problem infinite-dimensional.The convexity of p can be used to show that when the function α_i is taken to be a constant function for every i, a solution to problem (3.1) can be found by determining these constants. Before this can be proven, the following lemma is required.

Lemma 3.1 Let p be a convex function. A solution to

$$\begin{aligned}
\min_{\alpha,t} \quad & \int_0^{t} p(\alpha(\tau))d\tau \\
\text{s.t.} \quad & \int_0^{t} \alpha(\tau)w^{(0)}d\tau = w
\end{aligned}$$

is the constant function $\alpha(\tau) = \frac{w}{tw^{(0)}}$. This solution is unique if p is strictly convex. □

Proof For p convex and a function $\alpha : \mathbb{R}^+ \to A$, Jensen's inequality (see Theorem A.2) can be used:

$$\int_0^1 p(\alpha(\tau))d\tau \ge p\left(\int_0^1 \alpha(\tau)d\tau\right).$$

By the substitution rule, this can be written as:

$$\frac{1}{t}\int_0^t p(\alpha(\tau))d\tau \ge p\left(\frac{1}{t}\int_0^t \alpha(\tau)d\tau\right).$$

Equality holds if and only if $\alpha(\tau)$ is constant or if p is linear. Then by Jensen's inequality and equation (1.1),

$$\begin{aligned}
\int_0^t p\left(\frac{w}{tw^{(0)}}\right)d\tau &= tp\left(\frac{w}{tw^{(0)}}\right) \\
&= tp\left(\frac{1}{tw^{(0)}}\int_0^t \alpha(\tau)w^{(0)}d\tau\right) \\
&\le \int_0^t p(\alpha(\tau))d\tau.
\end{aligned}$$

■

It follows from Jensen's inequality that the constant solution is the only solution if p is not linear.

From this, the next corollary immediately follows.

Corollary 3.1 If there is a solution to the infinite-dimensional optimisation problem (3.1), then there is a solution $((\alpha_n),(t_n))$ such that for all i, α_i is a constant function. When p is strictly convex, it must be the case that (α_n) are constant functions. □

PROOF Take an arbitrary solution $((\tilde{\alpha}_n),(t_n))$ to problem (3.1), this will be used to construct a new solution (α_n) with constant scaling functions for each task. For each task $i \in \{1,\dots,N\}$, the work w_i can be calculated using the solution $\tilde{\alpha}_i$ and t_i. A new constant solution α_i is desired for each task, such that the execution time t_i is the same, but the costs do not increase. A solution with the same costs and the same t_i can be found using

$$\begin{aligned} \min_{\alpha_i} \quad & \int_0^{t_i} p(\alpha_i(\tau))d\tau \\ \text{s.t.} \quad & \int_0^{t_i} w^{(0)}\alpha_i(t)dt = w_i \end{aligned}$$

and this problem has $\tilde{\alpha}_i$ as a solution. From Lemma 3.1 follows that $\alpha_i = \frac{w_i}{t_i w^{(0)}}$ is also a solution to this problem. This can be done for all $i \in \{1,\dots,N\}$. Hence, there is a solution (α_n) for which all functions α_i in (α_n) are constant functions, as was claimed. If p is strictly convex, it follows that all functions α_i in (α_n) are necessarily constant functions. ■

For ease of notation, since $\alpha_i(\tau)$ can be assumed to be a constant function, $\alpha_i \in [L,1]$ is used instead of $\alpha_i(\tau)$. Now α_i is referred to as the scaling factor. Since α_i is a constant, the constraint can be written as

$$\int_0^{t_i} w^{(0)}\alpha_i dt = w^{(0)}t_i\alpha_i = w_i.$$

This can be rewritten to $t_i = \frac{w_i}{w^{(0)}\alpha_i}$. By substituting this into the optimisation problem (3.1), this gives

$$\begin{aligned} \min_{\alpha_1,\dots,\alpha_N} \quad & \sum_{i=1}^{N} p(\alpha_i)\frac{w_i}{\alpha_i w^{(0)}} & \\ \text{s.t.} \quad & \sum_{i=1}^{n} \frac{w_i}{\alpha_i w^{(0)}} \le d_n & \text{for all } n \in \{1,\dots,N\}, \quad (3.2) \\ & \alpha_n - 1 \le 0 & \text{for all } n \in \{1,\dots,N\}, \\ & L - \alpha_n \le 0 & \text{for all } n \in \{1,\dots,N\}. \end{aligned}$$

For brevity it is assumed that $w^{(0)} = 1$, the method of finding the solution will not change because of this. By using $\bar{p}(\alpha_i) = \frac{p(\alpha_i)}{\alpha_i}$, the problem can be written as:

$$\begin{aligned}
\min_{\alpha_1,\ldots,\alpha_N} \quad & \sum_{i=1}^{N} \bar{p}(\alpha_i) w_i \\
\text{s.t.} \quad & \sum_{i=1}^{n} \frac{w_i}{\alpha_i} \leq d_n && \text{for all } n \in \{1,\ldots,N\}, \\
& \alpha_n - 1 \leq 0 && \text{for all } n \in \{1,\ldots,N\}, \\
& L - \alpha_n \leq 0 && \text{for all } n \in \{1,\ldots,N\}.
\end{aligned} \tag{3.3}$$

3.2 Algorithm for the finite-dimensional problem

This section discusses the solution of optimisation problem (3.3). Problem (3.3) is a non-linear convex optimisation problem.

Since the work (w_n) of all tasks influences the solution of problem (3.3), it is not possible to find an elegant expression for the solution (α_n). Instead an algorithm, namely Algorithm 3.1, is used to find a solution (α_n) of problem (3.3) (proven in Theorem 3.1). This algorithm has several favourable properties that will be informally discussed, before they are formally proven. The first iteration of this algorithm determines scaling factors for the first h tasks, where $\alpha_1 = \alpha_2 = \cdots = \alpha_h$. The finishing time of task h (the last task in the first iteration) coincides with the deadline of this task. For tasks $h+1,\ldots,N$, the same procedure can be repeated by using subsequent iterations of the algorithm.

By choosing the scaling factors by using Algorithm 3.1, the scaling factors will be non-increasing. The intuition behind this is that it is inefficient to run run a task at a low speed and increasing the speed for the following task. In that case it turns out to be better to run both tasks at a single speed (a weighted average the speeds of the two tasks), this is due to the convexity of $\bar{p}$. This is always possible, since the speed of the first task is increased.

These results will be formalised in Lemma 3.3, Lemma 3.2 and Lemma 3.4. In case $\bar{p}$ is strictly convex, it turns out that the solution that is produced by Algorithm 3.1 is unique.

```
$j := 1$
while $j \leq N$ do
    $h := \max\left(\arg\max_{\bar{h}\in\{j,\ldots,N\}} \sum_{i=j}^{\bar{h}} \frac{w_i}{d_{\bar{h}} - d_{j-1}}\right)$
    $\alpha_j := \alpha_{j+1} := \cdots := \alpha_h := \max\left(L, \sum_{i=j}^{h} \frac{w_i}{d_h - d_{j-1}}\right)$
    $j := h + 1$
end
```

Algorithm 3.1: Optimal scaling factors

It is best to illustrate the algorithm using an example.

Example 3.1 Given is a process with four tasks (i.e., $N = 4$), $L = 0$, $p(\alpha) = \alpha^3$, $(w_n) = (10, 12, 3, 4)$ and $(d_n) = (20, 40, 60, 80)$. The cumulative work up to task n, is the sum of the work of tasks $1, \dots, n$. For the given tasks, this is given by the sequence $(10, 22, 25, 29)$. In figure 3.1, time is plotted against the cumulative work. This type of figures are used in the literature (for instance [8]) since they can show the work, deadlines, execution times and scaling factors. The graph "Full speed" shows the cumulative work that has been done at a certain time instant, when the process runs at the maximum speed $\alpha = 1$. The dashed horizontal lines indicate the cumulative work w_1, $w_1 + w_2$, $w_1 + w_2 + w_3$ and $w_1 + w_2 + w_3 + w_4$. When the graph crosses through such line, the task is finished (all work for this task is done). These intersections are marked using a triangle. If a scaling factor lower than 1 would be used, the triangle will move to the right over the dashed line.

The deadlines in the figure are given with respect to the cumulative work. Take for instance task 3, it has as cumulative work 25 and the deadline 60. This means that in total 25 cumulative work has to be done before time instant 60. All deadlines are met when the process runs at the maximum speed ($\alpha = 1$). The graph "Optimal" is above the graph that indicates the deadline. If this is not the case at some point, at least one deadline cannot be not met. It should be noted that all tasks finish long before their respective deadline. Hence, it is possible to choose lower scaling factors to decrease the energy consumption.

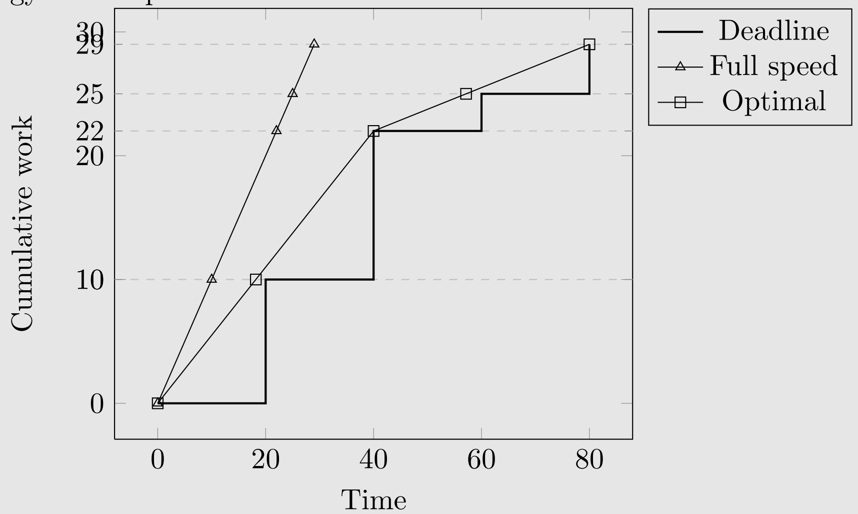


Figure 3.1: Demonstration of Algorithm 3.1

The graph "Optimal" shows the result of Algorithm 3.1. This result is used to illustrate how the algorithm works. In the first iteration of the algorithm, the value h is calculated as

$$h = \max\left(\arg\max_{\bar{h}\in\{1,\dots,4\}} \sum_{i=1}^{\bar{h}} \frac{w_i}{20\bar{h}}\right),$$

which corresponds to the highest cumulative work per time. In figure 3.1, it can be seen that the second task, with deadline 40, gives the maximum. Hence $h = 2$ and the resulting scaling factor is the slope of the graph "Optimal" between time 0 and 40. This scaling factor is used for task 1 and task 2.

The second iteration starts with task 3. The value h is calculated as

$$h := \max\left(\arg\max_{\bar{h}\in\{3,4\}} \sum_{i=3}^{\bar{h}} \frac{w_i}{20\bar{h} - 40}\right),$$

which gives $h = 4$. This implies that $\alpha_3 = \alpha_4$. The scaling factors again correspond to the slope of the graph.

Since the graph "Optimal" is above the graph that gives the deadlines, all deadlines are met.

Using this example, a few properties of the algorithm can be observed from the graph "Optimal", the graph that corresponds to the result of Algorithm 3.1. The scaling factor is non-increasing, this can be observed by looking at the slope of the graph. Task 2 is the last task that is used in the first iteration of the algorithm, while task 4 is the last task in the second iteration of the algorithm. It can be observed from figure 3.1 that these tasks meet the deadline exactly. Furthermore, the graph is increasing and above the graph "Deadline". This indicates that the solution is feasible. If any of the scaling factors is decreased without increasing other scaling factors, the solution will not be feasible anymore. As will be shown by Theorem 3.1, the result from the algorithm is even optimal.

The properties that were just described have to be proven. In order to do this, additional variables are introduced to make it easier to prove properties of this algorithm. This is done by extending Algorithm 3.1 to Algorithm 2. After running Algorithm 2, the variable $\bar{k}$ denotes the number of iterations of the while loop the algorithm went through. The values $H_1, \ldots, H_{\bar{k}}$ give the last task for each iteration. Hence, iteration 1 finds the scaling factors of tasks $H_0 + 1, \ldots, H_1$, iteration 2 finds the scaling factors of tasks $H_1 + 1, \ldots, H_2$, etc. This means that the first iteration of the while loop defines the scaling factors $\alpha_{H_0+1}, \ldots, \alpha_{H_1}$, the second iteration defines the scaling factors $\alpha_{H_1+1}, \ldots, \alpha_{H_2}$, etc.

In figure 3.2, some of these variables are shown. In this figure, $d_{H_0} = d_0$, $d_{H_1} = d_2$ and $d_{H_2} = d_4$. At the right side of the figure, the cumulative work is shown, while at the top the deadlines are given. The iterations are given using pairs of braces. The pair of braces for the first iteration indicate $d_{H_1} - d_{H_0}$ and $w_1 + w_2$. Note that $\alpha_1 = \alpha_2 = \frac{w_1+w_2}{d_{H_1}-d_{H_0}}$. Similarly, the second pair of braces show how the algorithm relates d_{H_1} and d_{H_2} to tasks 3 and 4.

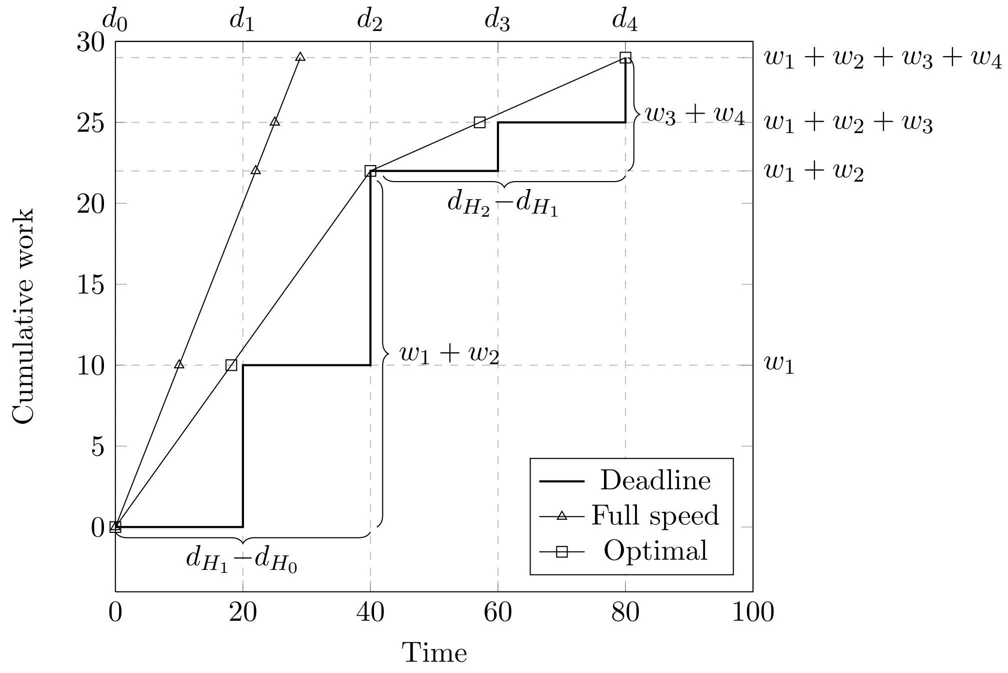


Figure 3.2: Demonstration of Algorithm 2

```
$j := 1$
$k := 1$
$H_0 := 0$
while $j \leq N$ do
    $h := \max\left(\arg \max_{\bar{h} \in \{j,\dots,N\}} \sum_{i=j}^{\bar{h}} \frac{w_i}{d_{\bar{h}} - d_{j-1}}\right)$
    $\alpha_j := \alpha_{j+1} := \cdots := \alpha_h := \max\left(L, \sum_{i=j}^{h} \frac{w_i}{d_h - d_{j-1}}\right)$
    $H_k := h$
    $j := h + 1$
    $k := k + 1$
end

$\bar{k} := k$
```

Algorithm 3.2: Optimal scaling factors, extended for proofs

Although the complexity of the algorithm is in worst case $O(N^2)$, namely maximally N iterations with in each iteration at most (a constant times) N steps (due to finding the maximum). In practice the complexity is approximately $O(N)$ since for many practical cases only a few iterations are required. Chapter 5 will discuss this in more detail. It was already observed that the scaling factors produced by Algorithm 3.1 are non-increasing. This is proven in the following lemma.

Lemma 3.2 The sequence (α_n) as generated by Algorithm 3.1 is a non-increasing sequence.

Proof Again consider Algorithm 2. Now this lemma is proven using proof by contradiction. Assume this lemma is not correct so that there is an α_n and α_{n+1} such that $\alpha_n < \alpha_{n+1}$. Since $\alpha_n \neq \alpha_{n+1}$, these values where produced by different iterations. Assume α_n was produced by iteration j. Then α_{n+1} was produced by iteration $j+1$ and $H_j = n$. First assume $\alpha_n \neq L$ and $\alpha_{n+1} \neq L$, then

$$\alpha_n = \sum_{i=H_{j-1}+1}^{H_j} \frac{w_i}{d_{H_j} - d_{H_{j-1}}} = \sum_{i=H_{j-1}+1}^{n} \frac{w_i}{n - d_{H_{j-1}}}$$

and

$$\alpha_{n+1} = \sum_{i=H_j+1}^{H_{j+1}} \frac{w_i}{d_{H_{j+1}} - d_{H_j}} = \sum_{i=n+1}^{H_{j+1}} \frac{w_i}{d_{H_{j+1}} - n}.$$

But,

$$\begin{aligned}
\alpha_n &> \sum_{i=d_{H_{j-1}}+1}^{H_{j+1}} \frac{w_i}{d_{H_{j+1}} - d_{H_{j-1}}} \\
&= \sum_{i=d_{H_{j-1}}+1}^{n} \frac{w_i}{d_{H_{j+1}} - d_{H_{j-1}}} + \sum_{i=n+1}^{H_{j+1}} \frac{w_i}{d_{H_{j+1}} - d_{H_{j-1}}} \\
&= \frac{d_{H_j} - d_{H_{j-1}}}{d_{H_{j+1}} - d_{H_{j-1}}}\alpha_n + \frac{d_{H_{j+1}-d_{H_j}}}{d_{H_{j+1}} - d_{H_{j-1}}}\alpha_{n+1} \\
&> \frac{d_{H_j} - d_{H_{j-1}}}{d_{H_{j+1}} - d_{H_{j-1}}}\alpha_n + \frac{d_{H_{j+1}-d_{H_j}}}{d_{H_{j+1}} - d_{H_{j-1}}}\alpha_n \\
&= \alpha_n
\end{aligned}$$

Which leads to a contradiction, hence $\alpha_n > \alpha_{n+1}$. In the case $\alpha_n = L$ and/or $\alpha_{n+1} = L$, it can be used that if

$$\sum_{i=H_{j-1}+1}^{H_j} \frac{w_i}{d_{H_j} - d_{H_{j-1}}} > \sum_{i=H_j+1}^{H_{j+1}} \frac{w_i}{d_{H_j} - d_{H_{j+1}}}$$

it is also true that

$$\max\left(L, \sum_{i=H_{j-1}+1}^{H_j} \frac{w_i}{d_{H_j} - d_{H_{j-1}}}\right) \geq \max\left(L, \sum_{i=H_j+1}^{H_{j+1}} \frac{w_i}{d_{H_j} - d_{H_{j+1}}}\right),$$

which shows that $\alpha_n \geq \alpha_{n+1}$. This is a contradiction. ■

The following lemma is used to make it easier to prove some important properties of Algorithm 3.1 and Algorithm 2. It shows that after each iteration the deadline is met exactly.

Lemma 3.3 Consider Algorithm 2

For all iterations $j \in \{1, \dots, \bar{k}\}$ there holds that either

$$\sum_{i=1}^{H_j} \frac{w_i}{\alpha_i} = d_{H_j}$$

or

$$\alpha_i = L \qquad \text{for all } i \in \{H_{j-1}+1, \dots, H_j\}.$$

Proof The lemma will now be proven using induction on j. The lemma is true for $j = 1$ (the basis step):

When $\sum_{i=1}^{H_1} \frac{w_i}{d_{H_1}-d_0} > L$,

$$\begin{aligned}\sum_{i=1}^{H_1} \frac{w_i}{\alpha_i} &= \sum_{i=1}^{H_1} \frac{w_i}{\sum_{l=1}^{H_1} \frac{w_l}{d_{H_1}-d_0}} \\ &= d_{H_1} - d_0 \\ &= d_{H_1}\end{aligned}$$

otherwise

$$\alpha_i = L \qquad \text{for all } i \in \{1, \dots, H_1\}.$$

Showing the basis step holds.

Now assume it is true for j, then for $j+1$ (induction step):

Either $\sum_{i=H_j+1}^{H_{j+1}} \frac{w_i}{\alpha_h} > L$ and (by Lemma 3.2) $\alpha_i > L$ for $1 \leq i \leq H_j$. Then

$$\begin{aligned}\sum_{i=1}^{H_{j+1}} \frac{w_i}{\alpha_h} &= \sum_{i=1}^{H_j} \frac{w_i}{\alpha_i} + \sum_{i=H_j+1}^{H_{j+1}} \frac{w_i}{\alpha_i} \\ &= d_{H_j} + \sum_{i=H_j+1}^{H_{j+1}} \frac{w_i}{\left[\frac{\sum_{l=H_j+1}^{H_{j+1}} w_l}{d_{H_{j+1}}-d_{H_j}}\right]} \\ &= d_{H_j} + d_{H_{j+1}} - d_{H_j} \\ &= d_{H_{j+1}},\end{aligned}$$

otherwise since $\alpha_i \geq L$,

$$\alpha_i = L \qquad \text{for all } i \in \{H_j+1, \dots, H_j\}.$$

Now the lemma follows by induction. ■

The last property of Algorithm 3.1 that was discussed, namely that it produces a feasible result, is discussed next.

Lemma 3.4 Algorithm 3.1 produces a feasible solution for optimisation problem (3.3).

Proof For all $n \in \{1, \dots, N\}$ is has to be shown that the constraints of problem (3.3) hold. For each n there exists a j, such that $H_j \leq n \leq H_{j+1}$. Then for the constraint $\alpha_n - 1 \leq 0$:

$$\begin{aligned}
&d_{H_j} + \sum_{i=H_j}^{n} w_i \leq d_n \\
\Rightarrow &\sum_{i=H_j}^{n} \frac{w_i}{d_n - d_{H_j}} \leq 1 \\
\Rightarrow &\max\left(L, \sum_{i=H_j}^{n} \frac{w_i}{d_n - d_{H_j}}\right) \leq 1 \\
\Rightarrow &\alpha_n \leq 1 \\
\Rightarrow &\alpha_n - 1 \leq 0,
\end{aligned}$$

where the first inequality is true since the process is admissible.

For the constraint $L - \alpha_n \leq 0$:

$$\begin{aligned}
&\alpha_n = \max\left(L, \sum_{i=d_{H_j}}^{H_{j+1}} \frac{w_i}{d_{H_{j+1}} - d_{H_h}}\right) \\
\Rightarrow &\alpha_n \geq L \\
\Rightarrow &L - \alpha_n \leq 0.
\end{aligned}$$

For the constraint $\sum_{i=1}^{n} \frac{w_i}{\alpha_i} \leq d_n$:

$$\begin{aligned}
\sum_{i=1}^{n} \frac{w_i}{\alpha_i} &= \sum_{i=1}^{H_j} \frac{w_i}{\alpha_i} + \sum_{i=H_j+1}^{n} \frac{w_i}{\alpha_i} \\
&= \sum_{i=1}^{H_j} \frac{w_i}{\alpha_i} + \sum_{i=H_j+1}^{n} \frac{w_i}{\sum_{l=H_j+1}^{n} \frac{w_l}{d_{H_{j+1}} - d_{H_j}}} \\
&\leq \sum_{i=1}^{H_j} \frac{w_i}{\alpha_i} + \sum_{i=H_j+1}^{n} \frac{w_i}{\sum_{l=H_j+1}^{n} \frac{w_l}{d_n - d_{H_j}}} \\
&= \sum_{i=1}^{H_j} \frac{w_i}{\alpha_i} + \left(d_n - d_{H_j}\right) \frac{\sum_{i=H_j+1}^{n} w_i}{\sum_{l=H_j+1}^{n} w_l} \\
&= d_{H_j} + d_n - d_{H_j} \\
&= d_n.
\end{aligned}$$

Where it is used that by Lemma 3.3, $\sum_{i=1}^{H_j} \frac{w_i}{\alpha_i} = d_{H_j}$ and it is used that $\sum_{l=H_j+1}^{H_{j+1}} \frac{w_l}{d_{H_{j+1}}-d_{H_j}} \geq \sum_{l=H_j+1}^{n} \frac{w_l}{d_n-d_{H_j}}$. ■

In iteration k of the while loop of the algorithm, a part of the problem is solved, namely from $H_{k-1}+1$ up to H_k inclusive. The scaling factors in this interval will be the same. The scaling factors for tasks $H_k+1,\dots,N$ will be lower than the scaling factor α_{H_k} or equal to L. In Lemma 3.3, it was shown that for each interval, the last task in this interval meets the deadline exactly or has the scaling factor L. In the proof of the next theorem, this is used, since the KKT point (when it is unique) apparently has this property.

Theorem 3.1 (optimal scaling factors) Algorithm 3.1 gives a solution of optimisation problem (3.3). □

Proof The cost function is (strictly) convex, since this sum of (strictly) convex functions is (strictly) convex by Lemma A.2. By Lemma A.9 the solution, if it exists, is unique. Since a KKT point exists, as will be shown, it is guaranteed that a solution exists. The feasible point given by Algorithm 3.1 will be a solution if the KKT conditions are satisfied:

$$\sum_{i=1}^{n} \frac{w_i}{\alpha_i} \leq d_n \qquad \text{for all } n \in \{1,\dots,N\}, \tag{3.4}$$

$$\alpha_n - 1 \leq 0 \qquad \text{for all } n \in \{1,\dots,N\}, \tag{3.5}$$

$$L - \alpha_n \leq 0 \qquad \text{for all } n \in \{1,\dots,N\}, \tag{3.6}$$

$$0 = w_n \bar{p}'(\alpha_n) - \frac{w_n}{\alpha_n^2}\left[\sum_{i=n}^{N} y_i\right] - z_n + q_n \qquad \text{for all } n \in \{1,\dots,N\}, \tag{3.7}$$

$$0 = y_n\left(\left[\sum_{i=1}^{n} \frac{w_i}{\alpha_i}\right] - d_n\right) \qquad \text{for all } n \in \{1,\dots,N\}, \tag{3.8}$$

$$0 = z_n(\alpha_n - 1) \qquad \text{for all } n \in \{1,\dots,N\}, \tag{3.9}$$

$$0 = q_n(L - \alpha_n) \qquad \text{for all } n \in \{1,\dots,N\}, \tag{3.10}$$

$$y_n \geq 0 \qquad \text{for all } n \in \{1,\dots,N\}, \tag{3.11}$$

$$z_n \geq 0 \qquad \text{for all } n \in \{1,\dots,N\}, \tag{3.12}$$

$$q_n \geq 0 \qquad \text{for all } n \in \{1,\dots,N\}. \tag{3.13}$$

The variables (y_n), (z_n) and (q_n) are dual variables. Now it will be shown that the dual variables together with the scaling factors (α_n) as produced by Algorithm 3.1 form a KKT point (a solution of the KKT conditions). Then it can be concluded from Theorem A.3 that (α_n) is a solution of problem (3.3). Because of Lemma 3.4, the conditions (3.4),(3.5) and (3.6) hold.

To check condition (3.7) for $n \in \{1,\dots,N\}$, a value l is defined. The value l is the highest value for which $\alpha_l \neq L$. If there is no such value, l is set to N. Hence:

$$l := \begin{cases} N & \text{if } L \notin \{\alpha_1,\dots,\alpha_N\}; \\ \min(\arg\min_{n\in\{1,\dots,N\}} \alpha_n) - 1 & \text{otherwise} \end{cases}$$

Because of lemma 3.2, if $i \geq l$, either $\alpha_i = L$ or $i = N$. For the remainder of the proof, (y_n), (q_n) and (z_n) are required for $1 \leq n \leq N$. Define these sequences as:

$$y_n := \begin{cases} \alpha_n^2 \bar{p}'(\alpha_n) - \alpha_{n+1}^2 \bar{p}'(\alpha_{n+1}) & \text{if } n < l; \\ \alpha_l^2 \bar{p}'(\alpha_l) & \text{if } n = l; \\ 0 & \text{if } n > l, \end{cases}$$

$$q_n := 0,$$

and

$$z_n := \begin{cases} w_n \bar{p}'(L) & \text{if } \alpha_n = L; \\ 0 & \text{otherwise.} \end{cases}$$

Now condition (3.7) will be checked. In case $\alpha_n > L$, then either $n < l$ and:

$$\begin{aligned} & w_n \bar{p}'(\alpha_n) - \frac{w_n}{\alpha_n^2}\left[\sum_{i=n}^{N} y_i\right] - z_n + q_n \\ &= w_n \bar{p}'(\alpha_n) - \frac{w_n}{\alpha_n^2}\left[\sum_{i=n}^{l-1} y_i\right] - \frac{w_n}{\alpha_n^2}\left[\sum_{i=l}^{N} y_i\right] \\ &= w_n \bar{p}'(\alpha_n) - \frac{w_n}{\alpha_n^2}\left[\sum_{i=n}^{l-1} \alpha_i^2 \bar{p}'(\alpha_i) - \alpha_{i+1}^2 \bar{p}'(\alpha_{i+1})\right] - \frac{w_n}{\alpha_n^2}\left[\alpha_l^2 \bar{p}'(\alpha_l)\right] \\ &= w_n \bar{p}'(\alpha_n) - \frac{w_n}{\alpha_n^2}\left[\alpha_n^2 \bar{p}'(\alpha_n) - \alpha_l^2 \bar{p}'(\alpha_l)\right] - \frac{w_n}{\alpha_n^2}\left[\alpha_l^2 \bar{p}'(\alpha_l)\right] \\ &= w_n \bar{p}'(\alpha_n) - w_n \bar{p}'(\alpha_n^2) + \frac{w_n}{\alpha_n^2}\left[\alpha_l^2 \bar{p}'(\alpha_l)\right] - \frac{w_n}{\alpha_n^2}\left[\alpha_l^2 \bar{p}'(\alpha_l)\right] \\ &= 0. \end{aligned}$$

or $n = l$ and:

$$\begin{aligned} & w_n \bar{p}'(\alpha_n) - \frac{w_n}{\alpha_n^2}\left[\sum_{i=n}^{N} y_i\right] - z_n + q_n \\ &= w_n \bar{p}'(\alpha_n) - \frac{w_n}{\alpha_n^2}\left[\sum_{i=l}^{l} y_i\right] \\ &= w_n \bar{p}'(\alpha_n) - \frac{w_n}{\alpha_n^2}\left[\alpha_l^2 \bar{p}'(\alpha_l)\right] \\ &= 0. \end{aligned}$$

In case $\alpha_n = L$, then $n > l$ and then:

$$\begin{aligned}
&w_n\bar{p}'(\alpha_n) - \frac{w_n}{\alpha_n^2}\left[\sum_{i=n}^{N} y_i\right] - z_n + q_n \\
&= w_n\bar{p}'(L) - w_n\bar{p}'(L) \\
&= 0.
\end{aligned}$$

This shows condition (3.7) holds. Now it will first be proven that condition (3.11) holds, since parts of the following proof will also be used to check condition (3.8). For $n \geq l$ it can be readily verified that condition (3.11) holds. To prove condition(3.11) holds for $n < l$, by Lemma 3.2:

$$\begin{aligned}
y_n &= \alpha_i^2\bar{p}'(\alpha_i) - \alpha_{i+1}^2\bar{p}'(\alpha_{i+1}) \\
&\geq \alpha_{i+1}^2\bar{p}'(\alpha_{i+1}) - \alpha_{i+1}^2\bar{p}'(\alpha_{i+1}) \\
&= 0.
\end{aligned}$$

Here it is used that $\bar{p}'(\alpha)$ is non-negative ($\bar{p}(\alpha)$ is non-decreasing) on $[L, 1]$ and since $\bar{p}(\alpha)$ is a convex function on $[L, 1]$, $\alpha^2\bar{p}'(\alpha)$ is an increasing function on $[L, 1]$. When $y_n > 0$ and $y_n = \alpha_n^2\bar{p}'(\alpha_n) - \alpha_{n+1}^2\bar{p}'(\alpha_{n+1})$, it follows that $\alpha_n > \alpha_{n+1}$ for the exact same reason.

To prove that condition (3.8) holds for $n \in \{1, \ldots, N\}$, first assume $y_n \neq 0$ because otherwise it holds trivially. When $y_n = \alpha_n^2\bar{p}'(\alpha_n)$, $n = l$. Then either $n = N$ and task n is the last task of some iteration. Or $\alpha_n > \alpha_{n+1} = L$, then also task n is the last task of some iteration. When $y_n = \alpha_n^2\bar{p}'(\alpha_n) - \alpha_{n+1}^2\bar{p}'(\alpha_{n+1})$, $\alpha_n > \alpha_{n+1}$ and task n is the last task of some iteration. In all cases, task n is the last task of some iteration, assume this is iteration j, and then $n = H_j$. Then by Lemma 3.3 and $\alpha_n \neq L$:

$$\begin{aligned}
\sum_{i=1}^{H_j} \frac{w_i}{\alpha_i} &= d_{H_j} \\
&= d_n.
\end{aligned}$$

Hence condition (3.8) holds. Condition (3.10) and condition (3.13) trivially hold, since $q_n = 0$ for all $n \in \{1, \ldots, N\}$. It can be readily verified that condition (3.12) holds. Condition (3.9) holds, since $z_n = 0$ for all n. ∎

It is interesting to see that the optimal solution does not depend on p. However, properties of p (convexity) and $\bar{p}$ (convexity, non-decreasing, etc.), are used to determine the solution. For instance, for the proof of theorem 3.1, it was used that $\bar{p}$ is convex and increasing. For Corollary 3.1 it was used that p is convex.

To demonstrate the algorithm, it will be written out for $L = 0$ and $N = 2$.

Example 3.2 Consider the case $L = 0$, $w^{(0)} = 1$, $N = 2$ with $d_1 = D$ and $d_2 = 2D$. It is assumed the process is admissible, hence $w_1 \leq D$ and $w_2 \leq D$. Then

$$\frac{w_1}{\alpha_1} \leq D \Leftrightarrow \frac{w_1}{D} \leq \alpha_1 \tag{3.14}$$

and $0 < \frac{w_1}{D} \leq 1$ since $w_1 > 0$, $D > 0$ and $w_1 \leq D$. Similarly

$$\frac{w_1}{\alpha_1} + \frac{w_2}{\alpha_2} \leq 2D \Leftrightarrow \frac{w_2}{2D - \frac{w_1}{\alpha_1}} \leq \alpha_2 \tag{3.15}$$

and $0 < \frac{w_2}{2D - \frac{w_1}{\alpha_1}} \leq 1$. This shows that the feasible region $\mathcal{F}$ is given by

$$\mathcal{F} = \left\{ (\alpha_1, \alpha_2) \middle| \frac{w_1}{D} \leq \alpha_1 \leq 1, \frac{w_2}{2D - \frac{w_1}{\alpha_1}} \leq \alpha_2 \leq 1 \right\}. \tag{3.16}$$

For $N = 2$, only two cases have to be considered.

(i) $w_1 \leq w_2$

For $j = 1$:

$$h = \max \arg \max_{\bar{h} \in \{1,2\}} \sum_{i=1}^{\bar{h}} \frac{w_i}{d_{\bar{h}}}.$$

This gives $h = 2$, since

$$\frac{w_1 + w_2}{2D} \geq \frac{w_1 + w_1}{2D} = \frac{w_1}{D}.$$

It now follows that $\alpha_1 = \alpha_2 = \frac{w_1 + w_2}{2D}$ gives the solution.

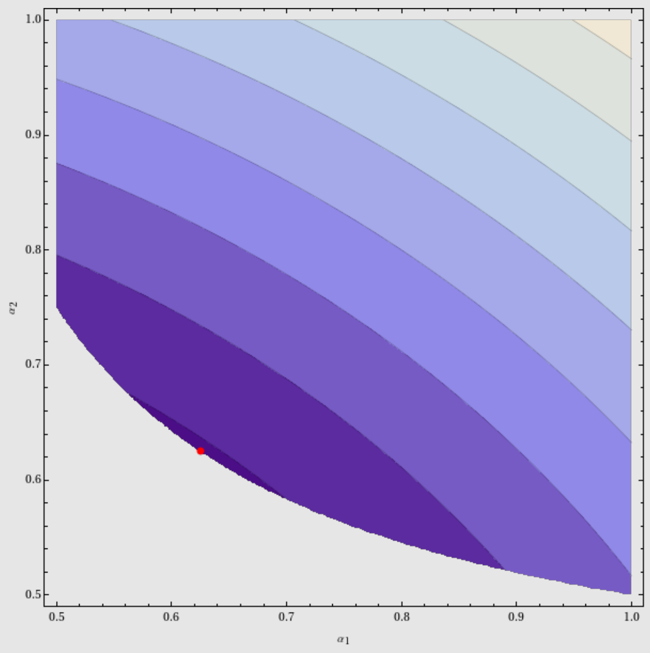


Figure 3.3: Optimisation problem (i)

In figure 3.3, a contour plot of the optimisation problem is shown where $w_1 \leq w_2$. The set of feasible solution $\mathcal{F}$ is shown in grey, where darker grey means less costs. The optimal solution is indicated with a red dot.

(ii) $w_2 \leq w_1$

For $j = 1$:

$$h = \max \arg \max_{\bar{h} \in \{1,2\}} \sum_{i=1}^{\bar{h}} \frac{t_i}{d_{\bar{h}}}.$$

This gives $h = 1$, since

$$\frac{w_1 + w_2}{2D} \leq \frac{w_1 + w_1}{2D} = \frac{w_1}{D}.$$

Then $\alpha_1 = \frac{w_1}{D}$. For $j = 2$:

$$h = \max \arg \max_{\bar{h} \in \{2\}} \frac{w_2}{D} = 2.$$

Then $\alpha_2 = \frac{w_2}{D}$.

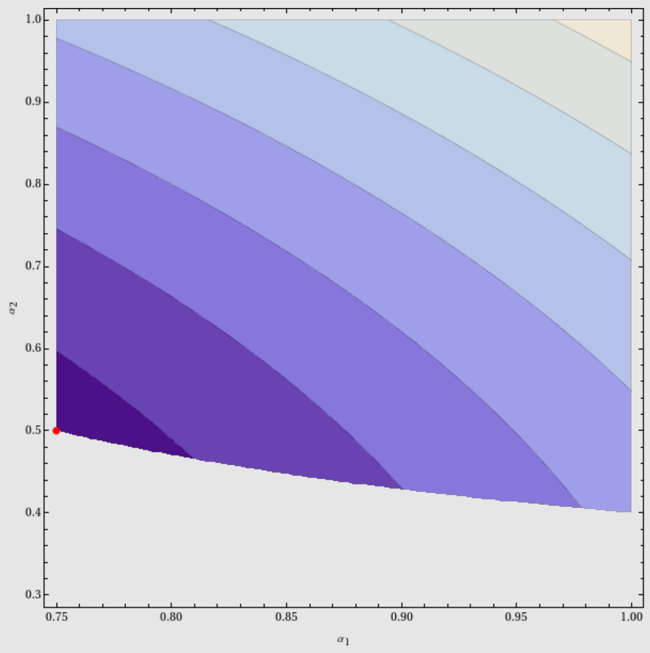


Figure 3.4: Optimisation problem (ii)

In figure 3.4, a contour plot of the optimisation problem is shown where $w_1 > w_2$. The optimal solution is again indicated with a red dot.

The next example will demonstrate two interesting properties of the algorithm. One property is that $\alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_N$ as was proven in Lemma 3.2. What will be observed in many examples is that consecutive scaling factors are often equal.

Example 3.3 Consider a process with $L = 0$, $D = 20$, $N = 9$, $(w_1, w_2, \ldots, w_9) = (10, 5, 7, 9, 8, 1, 7, 9, 10)$ and $(d_1, d_2, \ldots, d_9) = (D, 2D, \ldots, 9D)$.

For $j = 1$:

$$h = \max \arg \max_{\bar{h} \in \{1,\ldots,9\}} \sum_{i=1}^{\bar{h}} \frac{w_i}{\bar{h}D}.$$

It is easy to check (using Theorem 3.1) that $h = 1$, then $\alpha_1 = \frac{10}{20} = \frac{1}{2}$. For $j = 2$:

$$h = \max \arg \max_{\bar{h} \in \{2,\ldots,9\}} \sum_{i=2}^{\bar{h}} \frac{w_i}{\bar{h}D - D}.$$

This gives $h = 5$, then $\alpha_2 = \alpha_3 = \alpha_4 = \alpha_5 = \frac{29}{80}$. For $j = 3$:

$$h = \max\arg \max_{\bar{h} \in \{6,\ldots,9\}} \sum_{i=6}^{\bar{h}} \frac{w_i}{\bar{h}D - 5D}.$$

This gives $h = 9$, then $\alpha_6 = \alpha_7 = \alpha_8 = \alpha_9 = \frac{27}{80}$.

This shows that

$$(\alpha_1, \alpha_2, \ldots, \alpha_9) = \left(\frac{1}{2}, \frac{29}{80}, \frac{29}{80}, \frac{29}{80}, \frac{29}{80}, \frac{27}{80}, \frac{27}{80}, \frac{27}{80}, \frac{27}{80}\right).$$

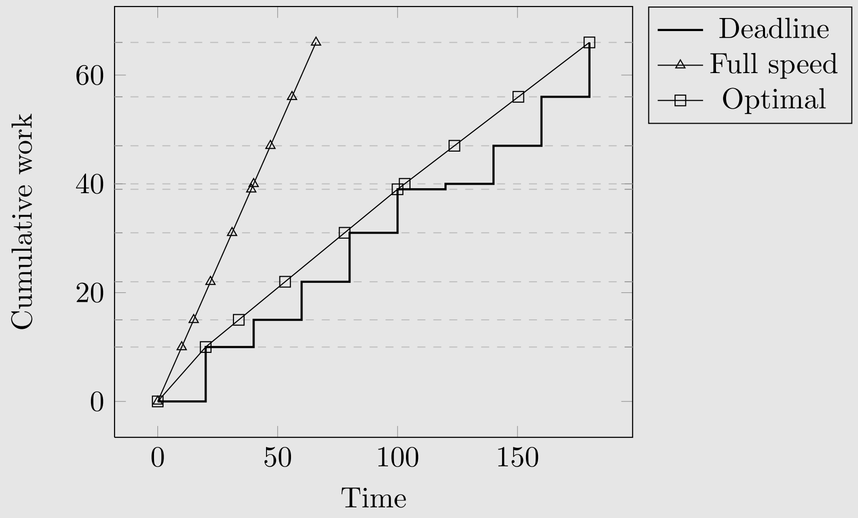


Figure 3.5: Deadlines and completion times after scaling

This solution is visualised in figure 3.5. The scaling factors form a non-increasing sequence, as was predicted by Lemma 3.2.

3.3 Restriction to finite set of scaling factors

Instead of $\alpha_i : [0, t_i] \to [L, 1]$, it is interesting to consider $\alpha_i : [0, t_i] \to A$ where $A \subset [L, 1]$ and $|A| = M$ (see definition 1.7. This is an important problem since many computer processors allow only for a finite set of scaling factors. Now Corollary 3.1 cannot be used anymore, since $\frac{w}{d}$ does not have to be in A. Now it will be shown how the infinite-dimensional problem (3.1) is reduced to a linear program in case A is finite.

The work w_n of task n is divided over the scaling factors $\bar{\alpha}_1, \ldots, \bar{\alpha}_M$. A fraction of the work w_n (namely $r_{n,1}$) is done at the speed given by scaling factor $\bar{\alpha}_1$, a different fraction of the work (namely $r_{n,2}$) is done at the speed given by scaling factor $\bar{\alpha}_2$, etc. Hence, $r_{n,m} \geq 0$ work is done by task n at the speed given by the scaling factor $\bar{\alpha}_m$ for a time duration $\frac{r_{n,m}}{\bar{\alpha}_m}$.

One can relate $\alpha_n(\tau)$ and $r_{n,m}$ as:

$$\alpha_n(\tau) := \begin{cases} \bar{\alpha}_1 & \text{if } \tau \leq \frac{r_{n,1}}{\alpha_1} \\ \bar{\alpha}_2 & \text{if } \frac{r_{n,1}}{\alpha_1} < \tau \leq \frac{r_{n,1}}{\alpha_1} + \frac{r_{n,2}}{\alpha_2} \\ \vdots & \vdots \\ \bar{\alpha}_M & \text{if } \sum_{j=1}^{M-1} \frac{r_{n,j}}{\alpha_j} < \tau. \end{cases}$$

Here the task begins with the speed given by the lowest scaling and ends with the highest scaling factor. This is an arbitrary choice, one can reorder the scaling factors, still the same work is done at the same costs.

Then the costs for task n are:

$$\int_0^{t_n} p(\alpha_n(\tau))d\tau = \sum_{j=1}^{M} p(\alpha_j)\frac{r_{n,j}}{\bar{\alpha}_j} = \sum_{j=1}^{M} \bar{p}(\bar{\alpha}_j) r_{n,j}.$$

and

$$\int_0^{t} \alpha_n(\tau) w^{(0)} d\tau = \sum_{j=1}^{M} \bar{\alpha}_j \frac{r_{n,j}}{\bar{\alpha}_j} = w_n$$

The sum of the work that is assigned to each scaling factor is the total work, i.e. $\sum_{m=1}^{M} r_{n,m} = w_n$. Then it follows immediately that the infinite-dimensional optimisation problem (3.1) can be reduced to to:

$$\begin{aligned} \min_{\substack{r_{i,j} \\ 1\leq i\leq N \\ 1\leq j\leq M}} \quad & \sum_{i=1}^{N}\sum_{j=1}^{M} r_{i,j}\bar{p}(\bar{\alpha}_j) \\ \text{s.t.} \quad & \sum_{i=1}^{n}\sum_{j=1}^{M} \frac{r_{i,j}}{\bar{\alpha}_j} \leq d_n & \text{for all } n \in \{1,\dots,N\}, \quad (3.17) \\ & \sum_{j=1}^{M} r_{n,j} = w_n & \text{for all } n \in \{1,\dots,N\}, \\ & r_{n,m} \geq 0 & \text{for all } n \in \{1,\dots,N\}, m \in \{1,\dots,M\}. \end{aligned}$$

Note that this is a linear program.

Instead of using a single function $\bar{p}$ that gives the energy per work, it is possible to replace this function by variables that give the energy per work for each task and for each scaling factor. Then $\bar{p}_{n,m}$ is the energy per work for task n and scaling factor m. This can be useful in case the power function does not only depend on α, but also on different variables. For instance, if the switched capacitance for all tasks are significantly different, this can be useful. See also section 1.2 and [9], for a discussion about this. Using the variables $\bar{p}_{m,n}$ instead of the function $\bar{p}(\alpha)$ as was used in problem (3.17) gives the following problem

$$\begin{aligned}
\min_{\substack{r_{i,j}\\ 1\le i\le N\\ 1\le j\le M}} \quad & \sum_{i=1}^{N}\sum_{j=1}^{M} r_{i,j}p_{i,j} \\
\text{s.t.} \quad & \sum_{i=1}^{h}\sum_{j=1}^{M} \frac{r_{i,j}}{\bar{\alpha}_j} \le d_n \quad \text{for all } n\in\{1,\dots,N\}, \\
& \sum_{j=1}^{M} r_{n,j} = w_n \qquad \text{for all } n\in\{1,\dots,N\}, r_{n,m}\ge 0 \quad \text{for all } n\in\{1,\dots,N\}, m\in\{1,\dots,M\}.
\end{aligned} \tag{3.18}$$

To solve this problem numerically, this form of the problem is not efficient. If it is written in matrix form, this results in a dense matrix, therefore it is not feasible to solve it for a big N, even on a modern computer. By introducing (f_n) (the completion times) as auxiliary variables, the problem can be reformulated as

$$\begin{aligned}
\min_{\substack{r_{i,j}\\ 1\le i\le N\\ 1\le j\le M}, \substack{f_i\\ 1\le i\le N}} \quad & \sum_{i=1}^{N}\sum_{j=1}^{M} r_{i,j}p_{i,j} \\
& f_n \le d_n \qquad \text{for all } n\in\{1,\dots,N\}, \\
& f_0 = 0, \\
& f_n = f_{n-1} + \sum_{j=1}^{M} \frac{r_{n,j}}{\bar{\alpha}_j} \quad \text{for all } n\in\{1,\dots,N\}, \\
& \sum_{j=1}^{M} r_{n,j} = w_n \qquad \text{for all } n\in\{1,\dots,N\}, r_{n,m}\ge 0 \quad \text{for all } n\in\{1,\dots,N\}, m\in\{1,\dots,M\}.
\end{aligned} \tag{3.19}$$

Example 3.4 Again, consider $(w_1, w_2, \dots, w_9) = (10, 5, 7, 9, 8, 1, 7, 9, 10)$, $D = 20$ and $(d_1, d_2, \dots, d_9) = (D, 2D, \dots, 9D)$. Now the scaling factors have to be chosen from the set $A = \{0.2, 0.5, 1\}$.

The linear problem (3.19) can be efficiently solved. This gives as solution

$$\begin{aligned}
(r_{1,1}, r_{1,2}, r_{1,3}, r_{2,1}, \dots) = (&0.0000, 10.0000, 0.0000, \\
&0.2590, 4.7410, 0.0000, \\
&0.4604, 6.5396, 0.0000, \\
&0.7289, 8.2711, 0.0000, \\
&1.2545, 6.7455, 0.0000, \\
&0.4930, 0.5070, 0.0000, \\
&2.5599, 4.4401, 0.0000, \\
&4.0187, 4.9813, 0.0000, \\
&6.2255, 3.7745, 0.0000).
\end{aligned}$$

This solution is feasible with the associated costs 13.14. However, this solution is not unique. For instance, consider

$$
\begin{aligned}
(r_{1,1}, r_{1,2}, r_{1,3}, r_{2,1}, \dots) = (&0, 10.0000, 0,\\
&1.2644, 3.7356, 0,\\
&1.7701, 5.2299, 0,\\
&2.2759, 6.7241, 0,\\
&2.0230, 5.9770, 0,\\
&0.3210, 0.6790, 0,\\
&2.2469, 4.7531, 0,\\
&2.8889, 6.1111, 0,\\
&3.2099, 6.7901, 0).
\end{aligned}
$$

This solution is still feasible and also has the costs 13.14, hence there is no unique optimal solution. Note that with

$$\tilde{\alpha}_i = \frac{w_i}{\sum_{j=1}^{M} \frac{r_{i,j}}{\bar{\alpha}_j}},$$

this is exactly the solution of Example 3.3. This relation is shown in figure 3.6 as the graph “Optimal (finite)”. In this graph it is shown in this solution, the scaling factors 0.2 and 0.5 are alternated. First, a part of the work is executed using scaling factor 0.2. The remainder of the work of a task is executed using scaling factor 0.5. This is the slope of the graph.

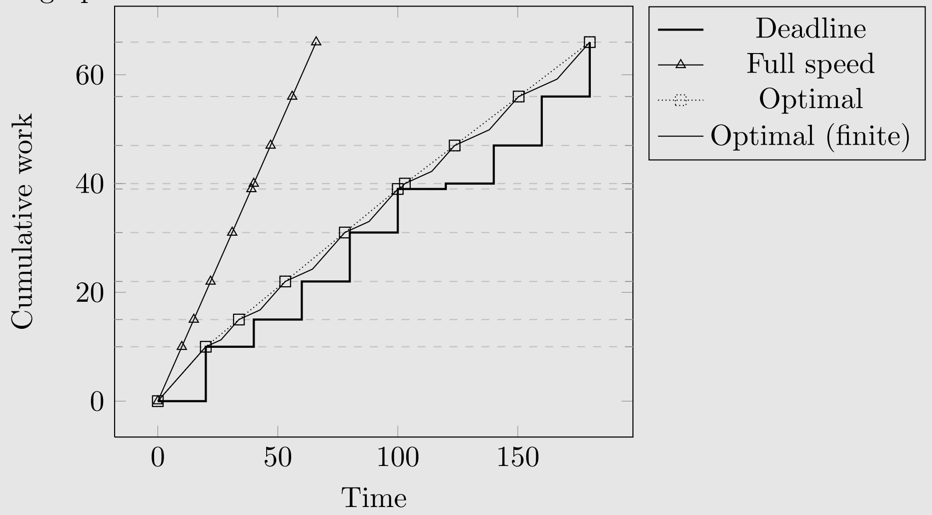


Figure 3.6: Finite set of scaling factors

The tasks of the given solution for a finite A are finished at the exact same time when $A = [0.2, 1]$ and Algorithm 3.1 is used. This suggests one can obtain a solution of problem (3.17) from the solution of problem (3.3). In the next section it will be shown that this is indeed the case.

This example shows that multiple solutions can exist. In fact, infinitely many solutions exist. One specific form of a solution is very useful. Assume there exists a solution given by $r_{i,j}$, for $i \in \{1, \dots, N\}$ and $j \in \{1, \dots, M\}$. Now define

Definition 3.1 (Total work) The total work $R_m \in \mathbb{R}^+$ that is done at a speed given by scaling factor $\bar{\alpha}_m \in \{1, \dots, M\}$ is given by

$$R_m := \sum_{i=1}^{N} r_{i,m} \qquad \text{for all } m \in \{1, \dots, N\}. \ \square$$

Instead of using the solution given by $r_{n,m}$ (for $n \in \{1, \dots, N\}$ and $m \in \{1, \dots, M\}$), it is possible to execute the first R_M amount of work at the highest speed α_M. The next R_{M-1} amount of work is executed at α_{M-1}, etc. In other words, the portion of the work that is executed at the highest scaling factor is moved to the beginning of the process, the same is done for the second highest scaling factor, etc. Naturally this new solution is still feasible and has the same costs. An advantage of a solution in this form is that the scaling factor changes at most $M - 1$ times during the execution of the entire process. When switching costs time or energy, this is clearly an advantage. This is illustrated in the following example.

Example 3.5 In example 3.4, it was shown that the solutions for $A = [0.2, 1]$ and $A = \{0.2, 0.5, 1\}$ are related, at least for this specific example. Now R_1, R_2 and R_3 will be calculated for this example:

$$R_1 = \sum_{i=1}^{N} r_{i,1} = 16$$
$$R_2 = \sum_{i=1}^{N} r_{i,2} = 50$$
$$R_3 = \sum_{i=1}^{N} r_{i,3} = 0.$$

Note that $R_1 + R_2 + R_3 = \sum_{i=1}^{N} w_i = 66$. Now for the first 50 work, the scaling factor 0.5 is used. For the remainder of the work (16), scaling factor 0.2 is used. This is shown in figure 3.7. From this figure, it is clear that the solution is feasible. Although this graph is above the graph of the other solution (for the same A), it is important to note that they both have the same costs. The exact costs are not visible in this graph.

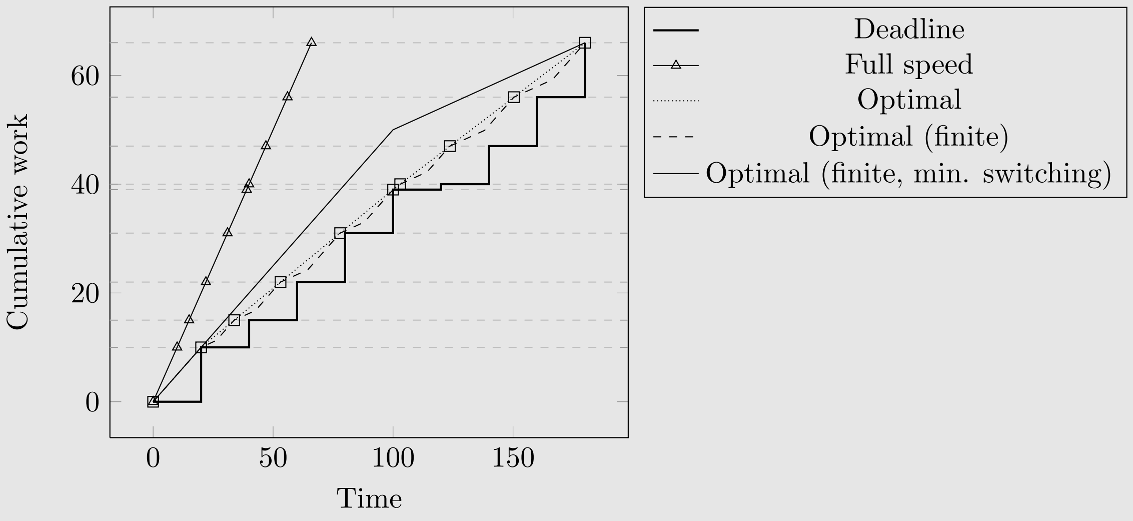


Figure 3.7: Minimal switching of optimal scaling factors

3.4 Relation between the continuous and the discrete problem

In Example 3.4 it was already suggested that problem (3.3) and problem (3.17) are related. Before the relation between problem (3.3) and problem (3.17) can be discussed, the following lemma is required.

Lemma 3.5 Given are $A = \{\bar{\alpha}_1, \dots, \bar{\alpha}_M\} \subset [L, 1]$ with $\bar{\alpha}_1 \leq \bar{\alpha}_2 \leq \cdots \leq \bar{\alpha}_M$ and the convex function $p : [L, 1] \to \mathbb{R}^+$. Then for all $m \in \{2, \dots, M-1\}$ and all $i \in \{1, \dots, m-1, m+1, \dots, M\}$:

$$p(\bar{\alpha}_i)(\bar{\alpha}_m - \bar{\alpha}_{m+1}) + p(\bar{\alpha}_{m+1})(\bar{\alpha}_i - \bar{\alpha}_m) + p(\bar{\alpha}_m)(\bar{\alpha}_{m+1} - \bar{\alpha}_i) \leq 0$$

Proof This proof is split up into the two cases $i < m$ and $i > m$. In case $i < m$:

Take $\lambda = \frac{\bar{\alpha}_m - \bar{\alpha}_{m+1}}{\bar{\alpha}_i - \bar{\alpha}_{m+1}}$, then $1 - \lambda = \frac{\bar{\alpha}_i - \bar{\alpha}_m}{\bar{\alpha}_i - \bar{\alpha}_{m+1}}$. Since $0 \leq \lambda \leq 1$ and $\bar{\alpha}_m = \lambda \bar{\alpha}_i + (1-\lambda)\bar{\alpha}_{m+1}$,

$$\begin{aligned} p(\bar{\alpha}_m) &= p(\lambda \bar{\alpha}_i + (1-\lambda)\bar{\alpha}_{m+1}) \\ &\leq \lambda p(\bar{\alpha}_i) + (1-\lambda) p(\bar{\alpha}_{m+1}) \\ &= \frac{\bar{\alpha}_m - \bar{\alpha}_{m+1}}{\bar{\alpha}_i - \bar{\alpha}_{m+1}} p(\bar{\alpha}_i) + \frac{\bar{\alpha}_i - \bar{\alpha}_m}{\bar{\alpha}_i - \bar{\alpha}_{m+1}} p(\bar{\alpha}_m). \end{aligned}$$

After multiplying by $\bar{\alpha}_i - \bar{\alpha}_{m+1}$, this gives:

$$\begin{aligned} & p(\bar{\alpha}_m)(\bar{\alpha}_i - \bar{\alpha}_{m+1}) \geq (\bar{\alpha}_m - \bar{\alpha}_{m+1}) p(\bar{\alpha}_i) + (\bar{\alpha}_i - \bar{\alpha}_m) p(\bar{\alpha}_m) \\ \Rightarrow & p(\bar{\alpha}_m)(\bar{\alpha}_{m+1} - \bar{\alpha}_i) + p(\bar{\alpha}_i)(\bar{\alpha}_m - \bar{\alpha}_{m+1}) + p(\bar{\alpha}_m)(\bar{\alpha}_i - \bar{\alpha}_m) \leq 0. \end{aligned}$$

which finishes the proof for $i < m$.

In case $i > m$:
Take $\lambda = \frac{\bar{\alpha}_{m+1}-\bar{\alpha}_i}{\bar{\alpha}_m-\bar{\alpha}_i}$, then $1-\lambda = \frac{\bar{\alpha}_m-\bar{\alpha}_{m+1}}{\bar{\alpha}_m-\bar{\alpha}_i}$. Note that $0 \leq \lambda \leq 1$ and $\bar{\alpha}_{m+1} = \lambda\bar{\alpha}_m + (1-\lambda)\bar{\alpha}_i$.

$$\begin{aligned} p(\bar{\alpha}_{m+1}) &= p(\lambda\bar{\alpha}_m + (1-\lambda)\bar{\alpha}_i) \\ &\leq \lambda p(\bar{\alpha}_m) + (1-\lambda)p(\bar{\alpha}_i) \\ &= \frac{\bar{\alpha}_{m+1}-\bar{\alpha}_i}{\bar{\alpha}_m-\bar{\alpha}_i}p(\bar{\alpha}_m) + \frac{\bar{\alpha}_m-\bar{\alpha}_{m+1}}{\bar{\alpha}_m-\bar{\alpha}_i}p(\bar{\alpha}_i) \end{aligned}$$

After multiplying by $\bar{\alpha}_m - \bar{\alpha}_i$, this gives:

$$\begin{aligned} & p(\bar{\alpha}_{m+1})(\bar{\alpha}_m-\bar{\alpha}_i) \geq p(\bar{\alpha}_m)(\bar{\alpha}_{m+1}-\bar{\alpha}_i) + p(\bar{\alpha}_i)(\bar{\alpha}_m-\bar{\alpha}_{m+1}) \\ \Rightarrow & p(\bar{\alpha}_{m+1})(\bar{\alpha}_i-\bar{\alpha}_m) + p(\bar{\alpha}_m)(\bar{\alpha}_{m+1}-\bar{\alpha}_i) + p(\bar{\alpha}_i)(\bar{\alpha}_m-\bar{\alpha}_{m+1}) \leq 0 \end{aligned}$$

which finishes the proof for $i > m+1$. ■

Now problem (3.17) is again considered. Although this linear problem can be solved analytically, it can also be solved by rewriting the result of Algorithm 3.1. This is discussed in the following theorem.

Theorem 3.2 Consider the following problem (see also problem 3.17).

$$\begin{aligned} & \min_{r_{i,j}} \sum_{i=1}^{N}\sum_{j=1}^{M} r_{i,j}\bar{p}(\bar{\alpha}_i) \\ & \text{s.t.} \sum_{i=1}^{n}\sum_{j=1}^{M} \frac{r_{i,j}}{\bar{\alpha}_i} \leq d_i && \text{for all } n \in \{n,\dots,N\} \\ & \sum_{j=1}^{M} r_{n,j} = w_n && \text{for all } n \in \{1,\dots,N\} \\ & r_{n,m} \geq 0 && \text{for all } n \in \{1,\dots,N\}, m \in \{1,\dots,M\} \end{aligned}$$

This can be solved by taking the solution (α_n) of Algorithm 3.1 with $L = \bar{\alpha}_1$. Then for all $n \in \{1,\dots,N\}$ there is an $l_n \in \{1,\dots,M\}$ such that

$$\bar{\alpha}_{l_n} \leq \alpha_n \leq \bar{\alpha}_{l_n+1}$$

and the optimal solution is then given by

$$r_{n,m} = \begin{cases} 0 & \text{if } m \neq l_n \wedge m \neq l_n+1 \\ w_n \frac{\frac{1}{\alpha_n}-\frac{1}{\bar{\alpha}_{l_n+1}}}{\frac{1}{\bar{\alpha}_{l_n}-\bar{\alpha}_{l_n+1}}} & \text{if } m = l_n; \\ w_n \frac{\frac{1}{\alpha_n}-\frac{1}{\bar{\alpha}_{l_n}}}{\frac{1}{\bar{\alpha}_{l_n+1}-\bar{\alpha}_{l_n}}} & \text{if } m = l_n+1 \end{cases}.$$

PROOF Since $\alpha_n \in [\bar{\alpha}_1, \bar{\alpha}_M]$, l_n exists for all $n \in \{1, \dots, N\}$.

The KKT conditions are given by

$$
\begin{aligned}
0 &\geq \sum_{i=1}^{n} \sum_{j=1}^{M} \frac{r_{i,j}}{\bar{\alpha}_i} - d_n && \text{for all } 1 \leq n \leq N, && (3.20)\\
0 &= \left[\sum_{j=1}^{M} r_{n,j} \right] - w_n && \text{for all } n \leq i \leq N, && (3.21)\\
0 &\leq r_{n,m} && \text{for all } 1 \leq n \leq N, 1 \leq m \leq M, && (3.22)\\
0 &= \bar{p}(\alpha_m) + \sum_{i=n}^{N} y_i \frac{1}{\bar{\alpha}_m} + \lambda_n - z_{n,m} && \text{for all } 1 \leq n \leq N, 1 \leq m \leq M, && (3.23)\\
0 &= z_{n,m} r_{n,m} && \text{for all } 1 \leq n \leq N, 1 \leq m \leq M, && (3.24)\\
0 &= y_n \left(\left[\sum_{i=1}^{n} \sum_{j=1}^{M} \frac{r_{i,j}}{\bar{\alpha}_j} \right] - d_n \right) && \text{for all } 1 \leq n \leq N, && (3.25)\\
0 &\leq y_n && \text{for all } 1 \leq n \leq N, && (3.26)\\
0 &\leq z_{n,m} && \text{for all } 1 \leq n \leq N, 1 \leq m \leq M. && (3.27)
\end{aligned}
$$

The solution is given by both $r_{i,j}$ as defined before and the dual variables

$$\bar{y}_n := \frac{p(\bar{\alpha}_{l_n+1}) - p(\bar{\alpha}_{l_n})}{\bar{\alpha}_{l_n+1} - \bar{\alpha}_{l_n}} \bar{\alpha}_{l_n} \bar{\alpha}_{l_n+1},$$

$$y_n := \begin{cases} \bar{y}_n - \bar{y}_{n+1} & \text{if } n < N; \\ 0 & \text{if } n = N, \end{cases}$$

$$\lambda_n := \frac{p(\bar{\alpha}_{l_n})\bar{\alpha}_{l_n} - p(\bar{\alpha}_{l_n+1})\bar{\alpha}_{l_n+1}}{\bar{\alpha}_{l_n+1} - \bar{\alpha}_{l_n}},$$

$$z_{n,m} = \begin{cases} 0 & \text{if } j = l_n \vee j = l_n + 1; \\ \bar{p}(\bar{\alpha}_m) + \bar{y}_n \frac{1}{\bar{\alpha}_m} + \lambda_n & \text{otherwise.} \end{cases}$$

To check condition (3.20):

$$\begin{aligned}
\sum_{i=1}^{n}\sum_{i=j}^{M}\frac{r_{i,j}}{\alpha_j} &= \sum_{i=1}^{n}\frac{r_{i,l_i}}{\alpha_{l_i}} + \frac{r_{i,l_i}}{\alpha_{l_i+1}} \\
&= \sum_{i=1}^{n}\left[\frac{w_i(\frac{1}{\alpha_i} - \frac{1}{\bar{\alpha}_{l_i+1}}\frac{1}{\bar{\alpha}_i})}{\frac{1}{\bar{\alpha}_i} - \frac{1}{\bar{\alpha}_{l_i+1}}} + \frac{w_i(\frac{1}{\alpha_i} - \frac{1}{\bar{\alpha}_{l_i}})\frac{1}{\alpha_{l_i+1}}}{\frac{1}{\bar{\alpha}_{l_i+1} - \frac{1}{\bar{\alpha}_{l_i}}}}\right] \\
&= \sum_{i=1}^{n} w_i \left[\frac{(\frac{1}{\alpha_i} - \frac{1}{\bar{\alpha}_{i_i+1}}) - (\frac{1}{\alpha_i} - \frac{1}{\bar{\alpha}_{l_i}})\frac{1}{\bar{\alpha}_{l_i+1}}}{\frac{1}{\bar{\alpha}_{l_i} - \frac{1}{\bar{\alpha}_{l_i+1}}}}\right] \\
&= \sum_{i=1}^{n}\frac{w_i}{\alpha_i} \\
&\leq d_n
\end{aligned}$$

To check condition (3.21), for all $n \in \{1, \ldots, N\}$:

$$\begin{aligned}
\sum_{j=1}^{M} r_{n,j} &= r_{n,l_n} + r_{n,l_n+1} \\
&= w_n\frac{\frac{1}{\alpha_n} - \frac{1}{\bar{\alpha}_{l_n+1}}}{\frac{1}{\bar{\alpha}_{l_n}} - \frac{1}{\bar{\alpha}_{l_n+1}}} + w_n\frac{\frac{1}{\alpha_n} - \frac{1}{\bar{\alpha}_{l_n}}}{\frac{1}{\bar{\alpha}_{l_n+1}} - \frac{1}{\bar{\alpha}_{l_n}}} \\
&= w_n\frac{\frac{1}{\alpha_n} - \frac{1}{\bar{\alpha}_{l_n+1}} - \frac{1}{\alpha_n} + \frac{1}{\bar{\alpha}_{l_n}}}{\frac{1}{\bar{\alpha}_{l_n}} - \frac{1}{\bar{\alpha}_{l_n+1}}} \\
&= w_n.
\end{aligned}$$

To check condition (3.22), for all $n \in \{1, \ldots, N\}$, there are three things to check. The case $j \neq l_n$ and $j \neq l_n + 1$ is trivially true. The cases $j = l_n$ and $j = l_n + 1$ are true since $\bar{\alpha}_{l_n} \leq \alpha_n \leq \bar{\alpha}_{l_n+1}$. Since conditions (3.20), (3.21) and (3.22) hold, the solution is feasible.

To check condition (3.23), this condition is rewritten to

$$0 = \bar{p}(\alpha_m) + \bar{y}_n\frac{1}{\bar{\alpha}_m} + \lambda_n - z_{n,m} \qquad \text{for all } 1 \leq n \leq N, 1 \leq m \leq M$$

and checked for $m = l_n$, $m = l_n + 1$ and $m \neq l_n \wedge m \neq l_n + 1$.

For $m = l_n$:

$$\begin{aligned}
\bar{p}(\bar{\alpha}_{l_n}) + \bar{y}_n\frac{1}{\bar{\alpha}_{l_n}} + \lambda_n &= \bar{p}(\bar{\alpha}_{l_n}) + \frac{\bar{p}(\bar{\alpha}_{l_n+1}) - \bar{p}(\bar{\alpha}_{l_n})}{\bar{\alpha}_{l_n+1} - \bar{\alpha}_{l_n}}\frac{\bar{\alpha}_{l_n}\bar{\alpha}_{l_n+1}}{\bar{\alpha}_{l_n}} + \frac{\bar{p}(\bar{\alpha}_{l_n})\bar{\alpha}_{l_n} - \bar{p}(\bar{\alpha}_{l_n+1})\bar{\alpha}_{l_n+1}}{\bar{\alpha}_{l_n+1} - \bar{\alpha}_{l_n}} \\
&= \bar{p}(\bar{\alpha}_{l_n}) + \frac{\bar{p}(\bar{\alpha}_{l_n})(\bar{\alpha}_{l_n} - \bar{\alpha}_{l_n+1})}{\bar{\alpha}_{l_n+1} - \bar{\alpha}_{l_n}} \\
&= \bar{p}(\bar{\alpha}_{l_n}) - \bar{p}(\bar{\alpha}_{l_n}) \\
&= 0.
\end{aligned}$$

For $m = l_n + 1$:

$$\begin{aligned}
\bar{p}(\bar{\alpha}_{l_n+1}) + \bar{y}_n \frac{1}{\bar{\alpha}_{l_n+1}} + \lambda_n &= \bar{p}(\bar{\alpha}_{l_n+1}) + \frac{\bar{p}(\bar{\alpha}_{l_n+1}) - \bar{p}(\bar{\alpha}_{l_n})}{\bar{\alpha}_{l_n+1} - \bar{\alpha}_{l_n}} \frac{\bar{\alpha}_{l_n}\bar{\alpha}_{l_n+1}}{\bar{\alpha}_{l_n+1}} + \frac{\bar{p}(\bar{\alpha}_{l_n})\bar{\alpha}_{l_n} - \bar{p}(\bar{\alpha}_{l_n+1})\bar{\alpha}_{l_n+1}}{\bar{\alpha}_{l_n+1} - \bar{\alpha}_{l_n}} \\
&= \bar{p}(\bar{\alpha}_{l_n+1}) + \frac{\bar{p}(\bar{\alpha}_{l_n+1})(\bar{\alpha}_{l_n} - \bar{\alpha}_{l_n+1})}{\bar{\alpha}_{l_n+1} - \bar{\alpha}_{l_n}} \\
&= \bar{p}(\bar{\alpha}_{l_n+1}) - \bar{p}(\bar{\alpha}_{l_n+1}) \\
&= 0.
\end{aligned}$$

For $m \neq l_n \wedge m \neq l_n + 1$:

$$\bar{p}(\bar{\alpha}_m) + \bar{y}_n \frac{1}{\bar{\alpha}_m} + \lambda_n - z_{n,m} = 0.$$

This follows directly from the definition of $z_{n,m}$.

Condition (3.24) holds, since by definition either $z_{n,m} = 0$ or $r_{n,m} = 0$.

Condition (3.25) is trivially true when $n = N$. For $n < N$:

$$\begin{aligned}
0 &= y_n \left(\left[\sum_{i=1}^{n} \sum_{j=1}^{M} \frac{r_{i,j}}{\bar{\alpha}_j} \right] - d_n \right) \\
&= (\bar{y}_n - \bar{y}_{n+1}) \left(\left[\sum_{i=1}^{n} \frac{w_i}{\alpha_i} \right] - d_n \right) \\
&= \left(\frac{\bar{p}(\bar{\alpha}_{l_n+1}) - \bar{p}(\bar{\alpha}_{l_n})}{\bar{\alpha}_{l_n+1} - \bar{\alpha}_{l_n}} \bar{\alpha}_{l_n+1}\bar{\alpha}_{l_n} - \frac{\bar{p}(\bar{\alpha}_{l_{n+1}+1}) - \bar{p}(\bar{\alpha}_{l_{n+1}})}{\bar{\alpha}_{l_{n+1}+1} - \bar{\alpha}_{l_{n+1}}} \bar{\alpha}_{l_{n+1}+1}\bar{\alpha}_{l_{n+1}} \right) \left(\left[\sum_{i=1}^{n} \frac{w_i}{\alpha_i} \right] - d_n \right).
\end{aligned}$$

If $\alpha_n = \alpha_{n+1}$ also $l_n = l_{n+1}$ and the first term is zero. If $\alpha_n \neq \alpha_{n+1}$, by Lemma 3.3 the second term is zero.

Condition (3.26) holds if $y_n \geq 0$. For $n = N$, this is trivially true. For $n < N$:

$$\begin{aligned}
y_n &= \bar{y}_n - \bar{y}_{n+1} \\
&= \frac{\bar{p}(\bar{\alpha}_{l_n+1}) - \bar{p}(\bar{\alpha}_{l_n})}{\bar{\alpha}_{l_n+1} - \bar{\alpha}_{l_n}} \bar{\alpha}_{l_n+1}\bar{\alpha}_{l_n} - \frac{\bar{p}(\bar{\alpha}_{l_{n+1}+1}) - \bar{p}(\bar{\alpha}_{l_{n+1}})}{\bar{\alpha}_{l_{n+1}+1} - \bar{\alpha}_{l_{n+1}}} \bar{\alpha}_{l_{n+1}+1}\bar{\alpha}_{l_{n+1}} \\
&\geq 0.
\end{aligned}$$

For $l_n = l_{n+1}$ this is trivially true. Otherwise this follows from Lemma A.1 and $\bar{\alpha}_{l_n+1} \geq \bar{\alpha}_{l_n} \geq \bar{\alpha}_{l_{n+1}} \geq \bar{\alpha}_{l_{n+1}+1}$.

Now condition (3.27), $z_{n,m} \geq 0$, will be checked. If $m = l_n$ or $m = l_n + 1$, this is easily verified. Otherwise, by Lemma 3.5:

$$
\begin{aligned}
& z_{n,m} \geq 0 \\
\Leftrightarrow & \bar{p}(\bar{\alpha}_m) + \bar{y}_n \frac{1}{\bar{\alpha}_m} + \lambda_n \geq 0 \\
\Leftrightarrow & \bar{p}(\bar{\alpha}_m) + \frac{\bar{p}(\bar{\alpha}_{l_n+1}) - \bar{p}(\bar{\alpha}_{l_n})}{\bar{\alpha}_{l_n+1} - \bar{\alpha}_{l_n}} \frac{\bar{\alpha}_{l_n}\bar{\alpha}_{l_n+1}}{\bar{\alpha}_m} + \frac{\bar{p}(\bar{\alpha}_{l_n})\bar{\alpha}_{l_n} - \bar{p}(\bar{\alpha}_{l_n+1})\bar{\alpha}_{l_n+1}}{\bar{\alpha}_{l_n+1} - \bar{\alpha}_{l_n}} \geq 0 \\
\Leftrightarrow & \bar{p}(\bar{\alpha}_m)\bar{\alpha}_m + \frac{\bar{p}(\alpha_{l_n+1})\bar{\alpha}_{l_n}\bar{\alpha}_{l_n+1} - \bar{p}(\bar{\alpha}_{l_n})\bar{\alpha}_{l_n}\bar{\alpha}_{l_n+1} + \bar{p}(\bar{\alpha}l_n)\bar{\alpha}_{l_n}\bar{\alpha}_m - \bar{p}(\bar{\alpha}_{l_n+1})\bar{\alpha}_{l_n+1}\bar{\alpha}_m}{\bar{\alpha}_{l_n+1} - \bar{\alpha}_{l_n}} \geq 0 \\
\Leftrightarrow & \bar{p}(\bar{\alpha}_m)\bar{\alpha}_m(\bar{\alpha}_{l_n+1} - \bar{\alpha}_{l_n}) + \bar{p}(\bar{\alpha}_{l_n+1})\bar{\alpha}_{l_n+1}(\bar{\alpha}_{l_n} - \bar{\alpha}_m) + \bar{p}(\bar{\alpha}_{l_n})\bar{\alpha}_{l_n}(\bar{\alpha}_m - \bar{\alpha}_{l_n+1}) \geq 0 \\
\Leftrightarrow & p(\bar{\alpha}_m)(\bar{\alpha}_{l_n} - \bar{\alpha}_{l_n+1}) + p(\bar{\alpha}_{l_n+1})(\bar{\alpha}_m - \bar{\alpha}_{l_n}) + p(\bar{\alpha}_{l_n})(\bar{\alpha}_{l_n+1} - \bar{\alpha}_m) \leq 0
\end{aligned}
$$

and by Lemma 3.5 this holds. Now all conditions have been checked and the theorem holds. ■

An interesting property of the solution that is given by Theorem 3.2 is that for each task, only two scaling factors are used. The following, well known result, directly follows from this theorem.

Corollary 3.2 Assume that p is convex and $\bar{p}$ is convex and increasing, and A is finite and $N = 1$. Then the task can be executed by using only two scaling factors $\bar{\alpha}_m$ and $\bar{\alpha}_{m+1}$ such that, with respect to problem (3.17):

(i) The task finishes before its deadline

(ii) The energy consumption is minimal □

Chapter 4

Online energy optimisation

In many applications, the work is not known before the process is started. Consider, for instance, a live video streaming application. A video broadcaster transmits encoded video frames to a receiver which decodes these frames. The work that is required to decode a future video frame is unknown for live video, it has not been recorded yet. Offline optimisation cannot be used.

In this chapter online optimisation is discussed. When the tasks $1, \ldots, n-1$ have finished their work, only the scaling factors for tasks $n, \ldots, N$ have to be determined. Since online optimisation does not know the exact future, the decisions it takes are not perfect. In almost all cases, the task will finish too early, the time the task is early is called the slack time (s_n). When a task n is finished early, slack time is used to decrease the speed of future tasks to decrease the energy consumption.

In section 4.1, a model which uses predictions and slack time to decrease energy is discussed. The feasible region of the problem derived in this section is too small. This is intentional, since this makes it easier to compare the results with the literature. The size of the feasible region will be corrected in section 4.2. In section 4.3, online energy minimisation with a finite number of scaling factors is discussed.

4.1 Using predictions

This section presents a model for online optimisation. A similar model has been studied in [19]. In this paper, it is (pessimistically) assumed that the WCW can occur for every future task. This is taken into account in the contraints. This restriction decreases the size of the feasible region to such extent that not always a feasible solution can be found. In that case a heuristic is used to find a feasible scaling factor.

Several differences exist between the approach in this section and the approach from [19]. First, a stochastic approach is used in [19]. Second, the approach in this section guarantees that a feasible solution can always be found. As will be shown in section 4.2, the feasible region is still too small, too much freedom is removed from the model. Nevertheless, this model will be discussed, since that makes it possible to compare results between this approach and the approach that is presented in section 4.2.

In section 3.1, the infinite dimensional offline optimisation problem was discussed. This model cannot be directly used for online optimisation. In this section, the work (w_n) is replaced by the prediction of the work, ($\tilde{w}_n$).

Definition 4.1 (Predicted work $\tilde{w}$) The predicted work $\tilde{w}$ is a prediction of the work w. For task n, the predicted work is denoted by $\tilde{w}_n$. Ideally $\tilde{w}_n = w_n$. □

Now the costs for task k are given by

$$\int_0^{t_k} p(\alpha_k(\tau))d\tau,$$

with

$$\int_0^{t_k} w^{(0)}\alpha_k(\tau)d\tau = \tilde{w}_k$$

These are the costs in case the prediction is correct, for the costs this is a reasonable assumption. Note that this is similar to what was done in problem (3.1). In problem (3.1), the constraint $\sum_{i=1}^{k} t_i \leq d_k$ was used to ensure that the deadline of task k is met. This cannot be done for the online problem, since two differences between the online and the offline problem have to be taken into account. First, the slack time s_n can be used, the slack time can be added to the deadline as it gives tasks some additional time to finish their work. Second, if the prediction is not accurate, the deadline can be missed. To take this into account, the following relation is used:

$$\int_0^{\bar{t}_k} w^{(0)}\alpha_k(\tau)d\tau = W.$$

Here $\bar{t}_k$ is the longest time it can take to execute task k, if W work has to be done. It is (in the worst case) possible that $W - \tilde{w}_k$ more work has to be done than estimated. In that case, if the remaining work is done at full speed (i.e., $\alpha_k(\tau) = 1$ for $\tau > t_k$), it takes $\bar{t}_k - t_k$ time longer to complete the task then expected. This shows that even when the prediction is not correct and the upper bound of the work is attained, at higher costs, the deadline can still be met. When $\sum_{i=1}^{k} \bar{t}_i \leq d_k + s_n$, the deadline will be met, even if the prediction is not correct. Then the infinite dimensional problem for online optimisation is given by:

$$\begin{aligned}
\min_{\substack{\alpha_n(\tau),\ldots,\alpha_N(\tau)\\ t_n,\ldots,t_N\\ \bar{t}_n,\ldots,\bar{t}_N}} \quad & \sum_{i=n}^{N} \int_0^{t_i} p(\alpha_i(\tau))d\tau \\
\text{s.t.} \quad & \int_0^{t_k} w^{(0)}\alpha_k(\tau)d\tau = \tilde{w}_k && \text{for all } k \in \{n,\ldots,N\}, \quad (4.1)\\
& \int_0^{\bar{t}_k} w^{(0)}\alpha_k(\tau)d\tau = W \\
& d_{n-1} - s_n + \sum_{i=n}^{k} \bar{t}_i \leq d_k && \text{for all } k \in \{n,\ldots,N\}.
\end{aligned}$$

This problem can be simplified by eliminating the variables $\bar{t}_n, \ldots, \bar{t}_N$. Note that, since $\bar{t}_k \geq t_k$,

$$\begin{aligned} W &= \int_0^{\bar{t}_k} w^{(0)} \alpha_k(\tau) d\tau \\ &= \int_0^{t_k} w^{(0)} \alpha_k(\tau) d\tau + \int_{t_k}^{\bar{t}_k} w^{(0)} \alpha_k(\tau) d\tau \\ &= \tilde{w}_k + \int_{t_k}^{\bar{t}_k} w^{(0)} d\tau \\ &= \tilde{w}_k + (\bar{t}_k - t_k) w^{(0)}, \end{aligned}$$

hence

$$\bar{t}_k = \frac{W - \tilde{w}_k}{w^{(0)}} + t_k.$$

After substituting this in problem (4.1), this problem is reduced to

$$\begin{aligned} \min_{\substack{\alpha_n(\tau),\ldots,\alpha_N(\tau) \\ t_n,\ldots,t_N}} \quad & \sum_{i=n}^{N} \int_0^{t_i} p(\alpha_i(\tau)) d\tau \\ \text{s.t.} \quad & \int_0^{t_k} w^{(0)} \alpha_k(\tau) d\tau = \tilde{w}_k && \text{for all } k \in \{n, \ldots, N\}, \\ & \sum_{i=n}^{k} t_i \leq d_k - d_{n-1} + s_n - \sum_{i=n}^{k} \frac{W - \tilde{w}_i}{w^{(0)}} && \text{for all } k \in \{n, \ldots, N\}. \end{aligned} \tag{4.2}$$

This is very similar to problem (3.1). Using the exact same reduction as was used in chapter 3, this can be reduced to:

$$\begin{aligned} \min_{\alpha_n,\ldots,\alpha_N} \quad & \sum_{i=n}^{N} p(\alpha_i) \frac{\tilde{w}_i}{\alpha_i w^{(0)}} \\ \text{s.t.} \quad & \sum_{i=n}^{k} \frac{\tilde{w}_i}{\alpha_i w^{(0)}} \leq d_k - d_{n-1} + s_n - \sum_{i=n}^{k} \frac{W - \tilde{w}_i}{w^{(0)}} && \text{for all } k \in \{n, \ldots, N\}, \\ & \alpha_k - 1 \leq 0 && \text{for all } k \in \{n, \ldots, N\}, \\ & L - \alpha_k \leq 0 && \text{for all } k \in \{n, \ldots, N\}. \end{aligned} \tag{4.3}$$

For brevity, it is assumed that $w^{(0)} = 1$ and used that $\bar{p}(\alpha) = \frac{p(\alpha)}{\alpha}$. Then

$$\begin{aligned} \min_{\alpha_n,\ldots,\alpha_N} \quad & \sum_{i=n}^{N} \bar{p}(\alpha_i) \tilde{w}_i \\ \text{s.t.} \quad & \sum_{i=n}^{k} \frac{\tilde{w}_i}{\alpha_i} \leq d_k - d_{n-1} + s_n - \sum_{i=n}^{k} (W - \tilde{w}_i) && \text{for all } k \in \{n, \ldots, N\}, \\ & \alpha_k - 1 \leq 0 && \text{for all } k \in \{n, \ldots, N\}, \\ & L - \alpha_k \leq 0 && \text{for all } k \in \{n, \ldots, N\}. \end{aligned} \tag{4.4}$$

This is similar to problem (3.3). For this reason, Algorithm 3.1 can be used to find the solution of problem (4.4). This algorithm states that α_n can be determined by:

$$\alpha_n = \max_{\bar{h} \in \{n,\ldots,N\}} \sum_{i=n}^{\bar{h}} \frac{\tilde{w}_i}{d_{\bar{h}} - d_{n-1} + s_n - \sum_{i=n}^{k} (W - \tilde{w}_i)}$$

Since the prediction can be wrong, it can happen that $w_n > \tilde{w}_n$. It is guaranteed that the deadline can still be met, if the scaling factor is increased. At most $W - \tilde{w}_n$ extra work has to be done. While in reality $w_n - \tilde{w}_n$ extra work has to be done, although this is not known before executing the process. This extra work has to be done in $d_n - d_{n-1} + s_n - \frac{\tilde{w}_n}{\alpha_n}$ time. Hence this scaling factor for the remaining work can be set to

$$\frac{W - \tilde{w}_n}{d_n - d_{n-1} + s_n - \frac{\tilde{w}_n}{\alpha_n}}.$$

Online optimisation is demonstrated in the following example.

Example 4.1 Again consider example 3.1. Given is a process with four tasks (i.e., $N = 4$), $L = 0$, $p(\alpha) = \alpha^3$, $(w_n) = (10, 12, 3, 4)$ and $(d_n) = (20, 40, 60, 80)$. When the prediction is perfect, i.e. $(\tilde{w}_n) = (10, 12, 3, 4)$, the optimal scaling factors are given by $(\alpha_n) = (1, 0.6552, 0.3746, 0.1445)$. When it is assumed that the prediction is given by $(\tilde{w}_n) = (8, 12, 4, 4)$, the optimal scaling factors are given by $(\alpha_n) = (1, 0.6667, 0.4, 0.1404)$.

In case $p(\alpha) = \alpha^3$, the costs when running at full speed are 29. When offline optimisation is used, the costs are brought back to 6.8694. A simple online approach is using the slack greedily, this gives the scaling factors $(\alpha_n) = (1, 0.6667, 0.6250, 0.3788)$ with costs 17.0792. The optimal online approach with a perfect "prediction" of the scaling factors has the costs 15.6556, while the optimal online approach which uses the given predictions has the actual costs 16.3312.

These results are shown in figure 4.1. In this figure, it can be seen that the online optimisation approaches and the greedy approach yield similar results for the first two tasks. For the third task, the greedy approach consumes all slack, while the online optimisation approach saves a part of the slack for task 4. Because of this, the energy consumption of the online optimisation approach gives a better result than the greedy approach.

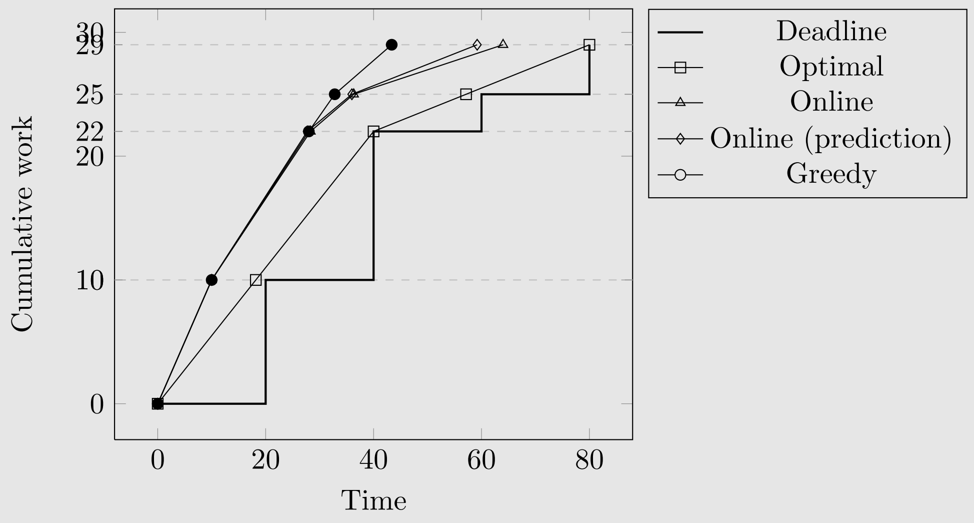


Figure 4.1: Demonstration of online optimisation

4.2 Loosening the constraints

In the previous section, an online optimisation model was discussed, where the constraints for the deadlines are very tight. Because the process is admissible, and if task n meets its deadline, there are always scaling factors $\alpha_{n+1}, \dots \alpha_N$ such that the deadlines for tasks $n{+}1, \dots, N$ can be met. This can be demonstrated using the definition of an admissible process. First, if task n meets its deadline, by the definition of an admissible process $d_n + \frac{w_n}{w^{(0)}} \leq d_{n+1}$. This shows that there is at least one scaling factor (e.g., $\alpha_n = 1$) by which the deadline can be met. This exact same reasoning can be used for tasks $n{+}2, \dots, N$. This demonstrates that assuming that the WCW can occur for future tasks $n{+}1, \dots, N$ is not required to guarantee that future deadlines can always be met. It is only required that task n meets its deadline.

Therefore the constraints for tasks $n{+}1, \dots, N$ in problem (4.4) can be loosened, i.e. the constraints that were introduced in the previous section are too tight. For these constraints, the WCW will not be considered anymore for tasks $n{+}1, \dots, N$. Now the problem can be written as

$$\begin{aligned}
\min_{\alpha_n,\dots,\alpha_N} \quad & \sum_{i=n}^{N} p(\alpha_i)\frac{\tilde{w}_i}{\alpha_i} \\
\text{s.t.} \quad & \frac{\tilde{w}_n}{\alpha_n} \leq d_n - d_{n-1} + s_n - (W - \tilde{w}_n) \\
& \sum_{i=n+1}^{k} \frac{\tilde{w}_i}{\alpha_i} \leq d_k - d_{n-1} + s_n && \text{for all } k \in \{n{+}1, \dots, N\}, \quad (4.5) \\
& \alpha_k - 1 \leq 0 && \text{for all } k \in \{n, \dots, N\}, \\
& L - \alpha_k \leq 0 && \text{for all } k \in \{n, \dots, N\}.
\end{aligned}$$

The feasible region becomes larger this way. For this reason, the costs decrease or remain the same, since there are more values where the minimum can be attained. Problem (4.5) can be solved using Algorithm 3.1.

The gains of loosening the constraints are demonstrated using the following example.

Example 4.2 Again consider example 3.1. Given is a process with four tasks (i.e., $N = 4$), $L = 0$, $p(\alpha) = \alpha^3$, $(w_n) = (10, 12, 3, 4)$ and $(d_n) = (20, 40, 60, 80)$.

In example 4.1, online optimisation was demonstrated using two different predictions. This will now be repeated in the context of problem (4.5). These results are shown in figure 4.2. In case of the perfect prediction $(\tilde{w}_n) = (10, 12, 3, 4)$, the optimal scaling factors are given by $(\alpha_n) = (1, 0.5455, 0.2727, 0.1905)$. This solution has as costs 13.9385. For the prediction $(\tilde{w}_n) = (8, 12, 4, 4)$, the optimal scaling factors are $(\alpha_n) = (1, 0.5455, 0.3333, 0.1739)$. For this solution, the costs are 14.0246.

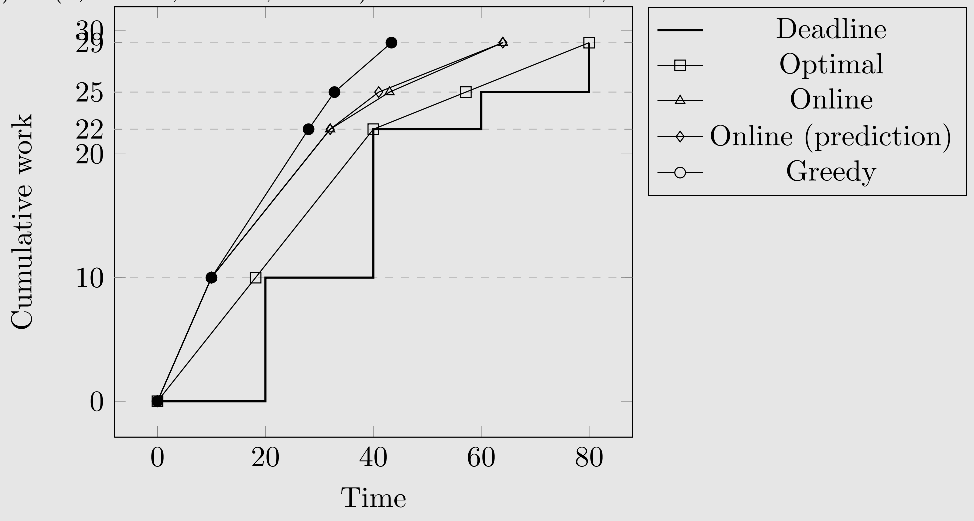


Figure 4.2: Demonstration of online optimisation

4.3 Restriction to a finite number of scaling factors

In section 3.3, the offline optimisation problem with a finite number of scaling factors to choose from was discussed. For online optimisation, the same will be done. Consider problem (4.2).

If only a finite number of scaling factors from the set $A = \{\bar{\alpha}_1, \ldots, \bar{\alpha}_M\}$ are allowed, this problem can be reduced to a linear program. In the exact same way as problem (3.1) was reduced to problem (3.17), problem (4.2) can be reduced to:

$$
\begin{aligned}
\min_{\substack{r_{i,j} \\ n\le i\le N \\ 1\le j\le M}} \quad & \sum_{i=n}^{N}\sum_{j=1}^{M} r_{i,j}\bar{p}(\bar{\alpha}_j) \\
\text{s.t.} \quad & \sum_{i=n}^{k}\sum_{j=1}^{M} \frac{r_{i,j}}{\bar{\alpha}_j} \le d_k - d_{n-1} + s_n - \sum_{i=n}^{k}(W-\tilde{w}_i) && \text{for all } k\in\{n,\dots,N\}, \\
& && (4.6) \\
& \sum_{j=1}^{M} r_{k,j} = \tilde{w}_k && \text{for all } k\in\{n,\dots,N\}, \\
& r_{k,m}\ge 0 && \text{for all } k\in\{n,\dots,N\}, m\in\{1,\dots,M\}.
\end{aligned}
$$

And by using the approach from section 4.2:

$$
\begin{aligned}
\min_{\substack{r_{i,j} \\ 1\le i\le N \\ 1\le j\le M}} \quad & \sum_{i=n}^{N}\sum_{j=1}^{M} r_{i,j}\bar{p}(\bar{\alpha}_j) \\
\text{s.t.} \quad & \sum_{j=1}^{M} \frac{r_{n,j}}{\bar{\alpha}_j} \le d_n - d_{n-1} + s_n - (W-\tilde{w}_n) && (4.7) \\
& \sum_{i=n}^{k}\sum_{j=1}^{M} \frac{r_{i,j}}{\bar{\alpha}_j} \le d_k - d_{n-1} + s_n && \text{for all } k\in\{n+1,\dots,N\}, \\
& && (4.8) \\
& \sum_{j=1}^{M} r_{k,j} = \tilde{w}_k && \text{for all } k\in\{n,\dots,N\}, \\
& r_{k,m}\ge 0 && \text{for all } k\in\{n,\dots,N\}, m\in\{1,\dots,M\}.
\end{aligned}
$$

This problem can be solved using Algorithm 3.1 together with Theorem 3.2. Hence, the solution of problem (4.7) can be found using the solution of problem (4.5).

Chapter 5

Evaluation

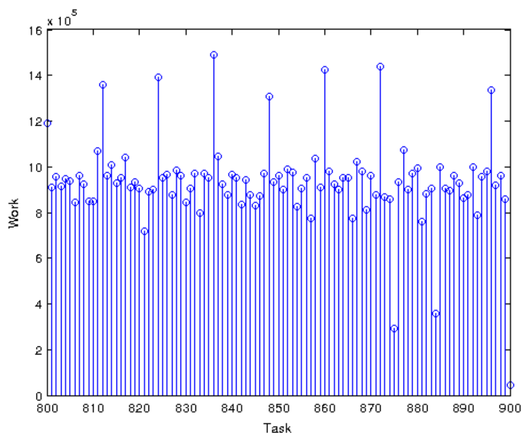
To evaluate the algorithms from chapter 3 and chapter 4, an MPEG-2 video decoder (as used in DVD players) is used as case study. Two DVD movies are used for the evaluation. The movies are decoded using *mpeg2dec*, an open source MPEG-2 decoder, on a PC with a 3GHz Intel Core 2 Duo processor. To make it easier to compare the movies, a video fragment of 45.000 frames are considered, which accounts for 30 minutes of video. The details on the video fragments can be found in table 5.1. It will be assumed that $w^{(0)} = 1$ and $p(\alpha) = \alpha^3$, to ease the calculations. The WCW is assumed to be the highest work per task (frame) encountered during the playback of the video fragment. To illustrate the variation in the work, the work of tasks 800 to 900 are shown in figure 5.1a for the first movie, while figure 5.1b shows the work for the second movie.

Algorithm 3.1 is used to calculate the optimal scaling factors for offline optimisation. The scaling factors are shown in figure 5.2a for DVD 1 and in figure 5.2b for DVD 2. Although the worst case complexity of Algorithm 3.1 is $O(N^2)$, the average case complexity is $O(N)$ since only a few frequency changes occur. Figures 5.2a and 5.2b confirm this. This shows that in practice, the algorithm is very efficient. In chapter 1 it was assumed that the overhead and costs of switching the clock frequency of a processor is low and can be ignored. Another reason why these costs can be ignored is that switching rarely occurs, even when a lot of clock frequencies are available.

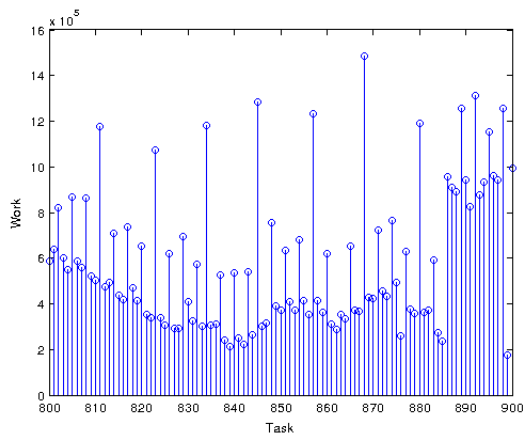
When only a few clock frequencies are available, more energy will be used, since the feasible region becomes smaller. Take for instance $A = \{0.05, 0.1, 0.2, 0.5, 1\}$. Then for DVD 1, only

	DVD 1	**DVD 2**
Title	The Shawshank Redemption	Up
File size	1GB	1GB
Subset of frames	2000-47000	2000-47000
Time duration	30 minutes	30 minutes
Minimum work	1073	1010
Maximum work	3896129	3472626
Average work	947617	648518

Table 5.1: Video fragements used for the evaluation

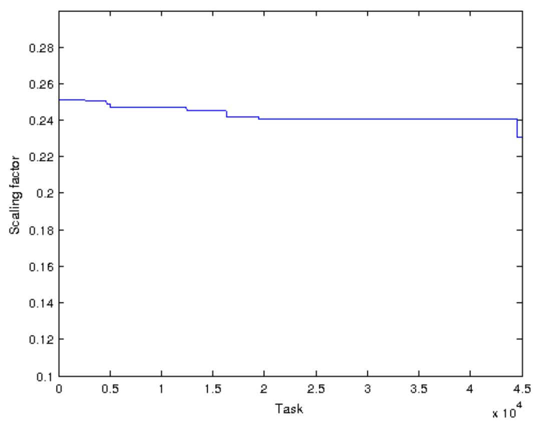


(a) DVD 1

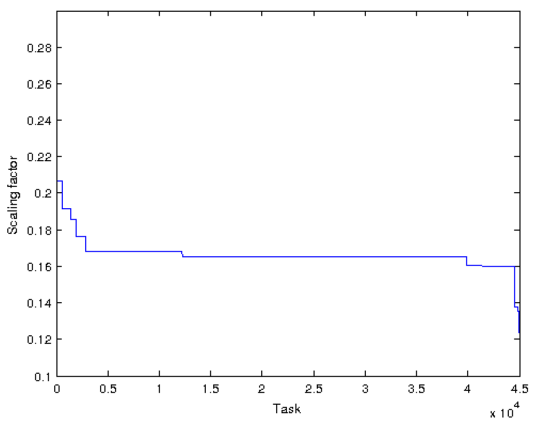


(b) DVD 2

Figure 5.1: Work for task 800 - 900



(a) DVD 1

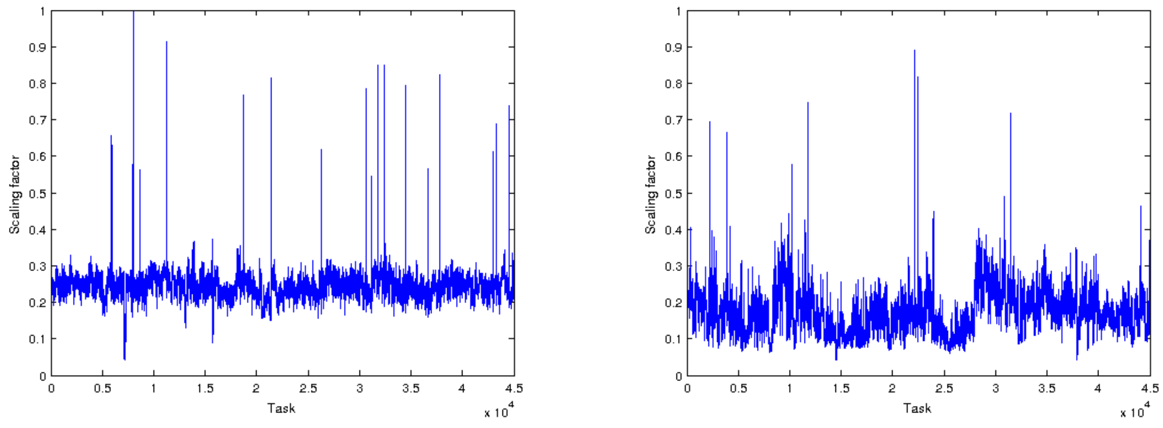


(b) DVD 2

Figure 5.2: Optimal scaling factors

Table 5.2: Optimisation characteristics of DVDs

	DVD 1	**DVD 2**
Energy consumption	4.2642×10^{10}	2.9183×10^{10}
Energy consumption, optimal (offline)	2.5245×10^{9}	0.8133×10^{9}
Energy consumption, finite set A	4.3579×10^{9}	0.9954×10^{9}
Energy consumption, greedy	2.6361×10^{9}	1.1209×10^{9}
Energy consumption, online (section 4.1)	8.6013×10^{9}	4.5874×10^{9}
Energy consumption, online (section 4.2)	2.5259×10^{9}	1.0244×10^{9}



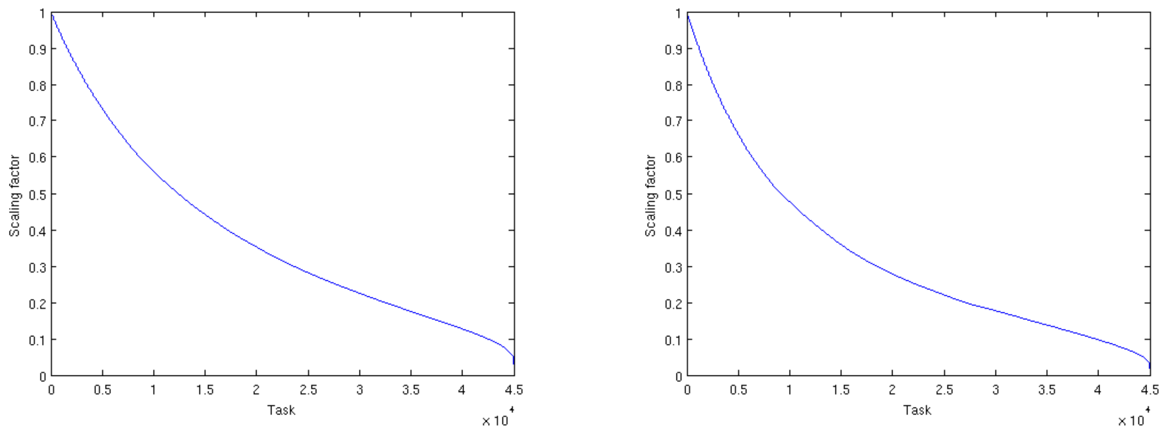
(a) DVD 1 (b) DVD 2

Figure 5.3: Scaling factors greedy algorithm

scaling factors 0.2 (α_2) and 0.5 (α_3) are used. Although by the result of Theorem 3.2 every task can run at two scaling factors, this solution is not unique. Similar to what is executed in section 3.3, the total work for all tasks that is done using scaling factor m, denoted by R_m is calculated for tasks $m \in \{1, \ldots, M\}$. This results in $R_1 = 0$, $R_2 = 0$, $R_3 = 3.0013 \times 10^{10}$, $R_4 = 1.2629 \times 10^{10}$ and $R_5 = 0$. It is possible to run the first tasks (until R_3 work is done) at speed 0.2 and after that all work is done at speed 0.5. Hence, the optimal is running for a time $\frac{R_3}{0.2}$ at speed 0.2 and at speed 0.5 for the remainder of the time. So only one switch of the clock frequency is required. For DVD 2 only clock frequencies 0.1, 0.2 and 0.5 are used.

To evaluate online optimisation, an (online) greedy algorithm (as introduced below) is used to compare the online optimisation from chapter 4 to. For the greedy algorithm, the scaling factor is set to $\frac{W}{D+s}$, hence all slack is immediately used. The scaling factors for DVD 1 and DVD 2 are shown in respectively figure 5.3a and figure 5.3b. The energy consumption is again shown in table 5.2. Surprisingly, the greedy algorithm is very efficient for this specific application. Compared to offline optimisation, only marginal gains are possible.

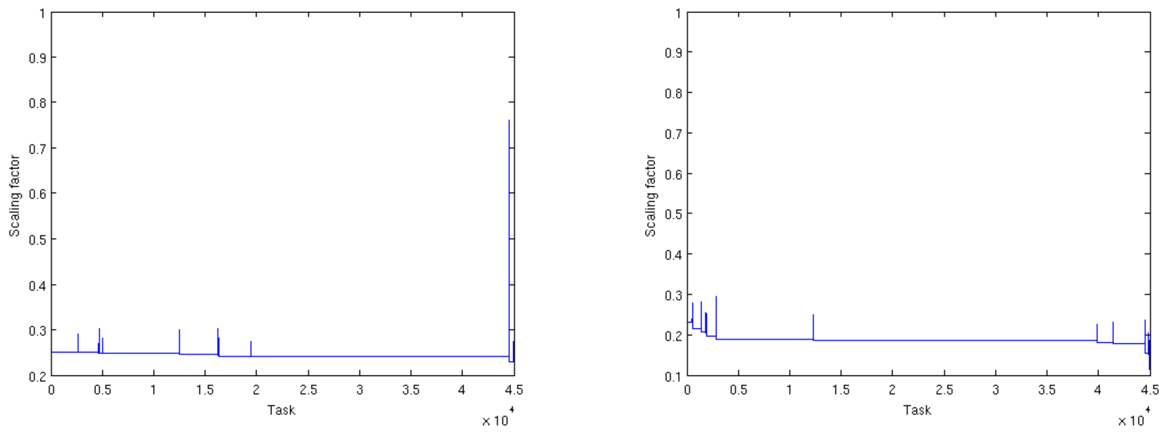
For online optimisation, the actual values are used instead of predictions (i.e., the predictions are perfect). The results of online optimisation as discussed in section 4.1 for DVD 1 and DVD 2 are shown in respectively figure 5.4a and figure 5.4b. The scaling factor starts



(a) DVD 1

(b) DVD 2

Figure 5.4: Scaling factors for online optimisation (section 4.1)



(a) DVD 1

(b) DVD 2

Figure 5.5: Scaling factors for online optimisation (section 4.2)

relatively high, since the algorithm tends to preserve slack for future tasks. However, in these two examples it starts too high (shown in figure 5.4a and figure 5.4b) and a lot energy is spent (shown in table 5.2). The energy consumption is much higher than when using the greedy approach.

The online optimisation problem as discussed in section 4.2 performs better. The scaling factors produced using this approach are shown in figure 5.5a and figure 5.5b for DVD 1 and DVD 2 respectively.

Table 5.2 shows that for DVD 1 and DVD 2, the energy consumption after offline optimisation and the energy consumption for the online approach from section 4.2 are very similar. The performance of online optimisation is not as good as the performance of offline optimisation as online optimisation has to wait for slack.

Chapter **6**

Future work

6.1 Nonuniform capacitances

In section 1.2, it is assumed the switched capacitance C is constant. In [9], the authors assume that each task has a switched capacitance C_i. This model is more accurate, but one has to measure the switched capacitance for each task. When switched capacitances are taken into account, problem (3.3) can be written as:

$$\begin{aligned} \min_{\alpha_1,\ldots,\alpha_N} \quad & \sum_{i=1}^{N} p(\alpha_i) C_i w_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} \frac{w_i}{\alpha_i} \leq d_n, && \text{for all } n \in \{1,\ldots,N\}, \quad (6.1) \\ & \alpha_n - 1 \leq 0 && \text{for all } n \in \{1,\ldots,N\}, \\ & L - \alpha_n \leq 0 && \text{for all } n \in \{1,\ldots,N\}. \end{aligned}$$

It is difficult (compared to problem (3.3)) to solve this problem. Solving this problem for a finite number of scaling factors can be done using an LP solver, as was suggested in [9]. However, an analytic solution is desired.

6.2 Online optimisation

The online energy optimisation approach as discussed in this thesis works well. However in practice, not the entire future can be predicted. For practical purposes, it is desirable to use only a single prediction. It would for instance be possible to use one prediction for the work of the next task to be executed, i.e. $\tilde{w}_n$. For $\tilde{w}_{n+1},\ldots,\tilde{w}_N$, the mean can be used as a prediction. It straightforward to calculate the optimal scaling factors using the theory presented in section 4.2. For future research, it should be studied how good the scaling factors are for practical data. The data that were used in chapter 5 can be used to evaluate these results.

A useful different approach is using a stochastic approach of online optimisation. In that case, the distribution of the work has to be determined.

6.3 Arrival times

In this thesis it was implicitly assumed that a task i can start if tasks $1, \ldots, i-1$ have finished. Depending on the application, it might be desired to consider the arrival times of tasks. If a task receives its data from a remote device, for example over a communication network, not all data is available when the first task begins. Therefore, the following definition is required.

Definition 6.1 (Arrival time) The arrival time of task i, $a_i \in \mathbb{R}^+$, is the earliest time at which task i is allowed to start. □

The offline optimisation problem (3.3) can be extended to take arrival times into account. Now the start times (b_n) are relevant.

$$\begin{aligned}
\min_{\alpha_1,\ldots,\alpha_N,b_1,\ldots,b_N} \quad & \sum_{i=1}^{N} p(\alpha_i)\frac{w_i}{w^{(0)}} & \\
\text{s.t.} \quad & b_{n-1} + \frac{w_{n-1}}{\alpha_{n-1}} \leq b_n & \text{for all } n \in \{1,\ldots,N\}, \\
& b_n + \frac{w_n}{\alpha_n} \leq d_n & \text{for all } n \in \{1,\ldots,N\}, \\
& a_n \leq b_n & \text{for all } n \in \{1,\ldots,N\}, \quad (6.2) \\
& \alpha_n - 1 \leq 0 & \text{for all } n \in \{1,\ldots,N\}, \\
& L - \alpha_n \leq 0 & \text{for all } n \in \{1,\ldots,N\}.
\end{aligned}$$

The constraint $b_{n-1}+\frac{w_{n-1}}{\alpha_{n-1}} \leq b_n$ enforces that task i cannot start before task $i-1$ is finished (i.e., the start time of task $i-1$ plus the time it takes to execute task $i-1$). Similarly, the constraint $b_n + \frac{w_n}{\alpha_n} \leq d_n$ makes sure that task n is to be finished before its deadline.

6.4 Finite buffers

The solutions of the offline optimisation problem can be such that some tasks are finished long before their respective deadlines. In practice, this means that their results have to be stored before they are used. For a video decoder, the decoded frame is not used before the deadline of the frame. If an (embedded) computer does not have a lot of memory to buffer results, it might be useful to take buffer sizes into account in the constraints of the optimisation problem. Problem (3.3) can be extended with constraints to enforce that at most R results are stored inside the buffers. This is the same as saying that for each task i, only tasks up to task $i+R-1$ are allowed to be finished before the deadline of task i. In the optimisation problem, this can be written as:

$$\begin{aligned}
\min_{\alpha_1,\ldots,\alpha_N} \quad & \sum_{i=1}^{N} p(\alpha_i) w_i \\
\text{s.t.} \quad & \sum_{i=1}^{n} \frac{w_i}{\alpha_i} \leq d_n, && \text{for all } n \in \{1,\ldots,N\}, \quad (6.3)\\
& \sum_{i=1}^{n} \frac{w_i}{\alpha_i} \geq d_{n-R+1} && \text{for all } n \in \{R,\ldots,N\},\\
& \alpha_n - 1 \leq 0 && \text{for all } n \in \{1,\ldots,N\},\\
& L - \alpha_n \leq 0 && \text{for all } n \in \{1,\ldots,N\}.
\end{aligned}$$

The constraint $-\sum_{i=1}^{n} \frac{w_i}{\alpha_i} \leq -d_{n-R+1}$ is used to ensure the buffers do not overflow. Since $g(\alpha) = -\sum_{i=1}^{n} \frac{w_i}{\alpha_i} + d_{n-R+1}$ is concave, the optimisation problem is not necessarily convex. Nevertheless, this is an important problem that should be studied in future research.

6.5 Multiple processors

In chapter 3, it was discussed how to minimise energy of a set of tasks running on a single processor. The given solution does not depend on the exact power function p, only on the properties of this power function as were given in section 1.4. It is desirable to generalise this result to multiple processors. This problem will not be formally introduced. Instead, it will be shown that (and why) this problem will be hard in general. The following example will illustrate this.

Example 6.1 Consider two tasks (referred to as producers), both producing a single result. A third task (referred to as the consumer) waits until both results are available. After this, the third task will execute. Just as in the previous chapter, an amount of work and a deadline is assigned to each task. In this example take $w_1 = w_2 = w_3 = 1$, $d_1 = d_2 = 20$, $L = 0$, $p(\alpha) = \alpha^q$, $q \geq 1$ and $d_3 = 25$. The non-linear program can be written as

$$\begin{aligned}
\min_{\alpha_1,\alpha_2,\alpha_3} \quad & \alpha_1^q + \alpha_2^q + \alpha_3^q \\
\text{s.t.} \quad & \frac{1}{\alpha_1} \leq 20 \\
& \frac{1}{\alpha_2} \leq 20 \\
& \frac{1}{\alpha_1} + \frac{1}{\alpha_3} \leq 25 \\
& \frac{1}{\alpha_2} + \frac{1}{\alpha_3} \leq 25 \\
& \alpha_1 \leq 1 \\
& \alpha_2 \leq 1 \\
& \alpha_3 \leq 1 \\
& -\alpha_1 \leq 0 \\
& -\alpha_2 \leq 0 \\
& -\alpha_3 \leq 0.
\end{aligned}$$

This non-linear program is convex. The KKT conditions, besides feasibility, are given by:

$$\begin{aligned}
0 = & \begin{bmatrix} q\alpha_1^{q-1} \\ q\alpha_2^{q-1} \\ q\alpha_3^{q-1} \end{bmatrix} + y_1 \begin{bmatrix} -\frac{1}{\alpha_1^2} \\ 0 \\ 0 \end{bmatrix} + y_2 \begin{bmatrix} 0 \\ -\frac{1}{\alpha_2^2} \\ 0 \end{bmatrix} + y_3 \begin{bmatrix} -\frac{1}{\alpha_1^2} \\ 0 \\ -\frac{1}{\alpha_3^2} \end{bmatrix} + y_4 \begin{bmatrix} 0 \\ -\frac{1}{\alpha_2^2} \\ -\frac{1}{\alpha_3^2} \end{bmatrix} \\
& + y_5 \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + y_6 \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + y_7 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + y_8 \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} + y_9 \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} + y_{10} \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \\
0 = & y_1 \left(\frac{1}{\alpha_1} - 20 \right) + y_2 \left(\frac{1}{\alpha_2} - 20 \right) + y_3 \left(\frac{1}{\alpha_1} + \frac{1}{\alpha_3} - 25 \right) + y_4 \left(\frac{1}{\alpha_2} + \frac{1}{\alpha_3} - 25 \right) \\
& + y_5 (\alpha_1 - 1) + y_6 (\alpha_2 - 1) + y_7 (\alpha_3 - 1) \\
& + y_8(-\alpha_1) + y_9(-\alpha_2) + y_{10}(-\alpha_3) \\
0 \leq & y_i \qquad \text{for all } i \in \{1, \dots, 10\}
\end{aligned}$$

Now assume $y_i = 0$ for $i \in \{1, 2, 5, 6, 7, 8, 9, 10\}$. This means that these constraints are not active.

Then

$$
\begin{aligned}
0 =& \begin{bmatrix} q\alpha_1^{q-1} \\ q\alpha_1^{q-1} \\ q\alpha_3^{q-1} \end{bmatrix} + y_3 \begin{bmatrix} -\frac{1}{\alpha_1^2} \\ 0 \\ -\frac{1}{\alpha_3^2} \end{bmatrix} + y_4 \begin{bmatrix} 0 \\ -\frac{1}{\alpha_1^2} \\ -\frac{1}{\alpha_3^2} \end{bmatrix} \\
0 =& y_3 \left(\frac{1}{\alpha_1} + \frac{1}{\alpha_2} - 25 \right) + y_4 \left(\frac{1}{\alpha_1} + \frac{1}{\alpha_3} - 25 \right) \\
0 \leq& y_3 \\
0 \leq& y_4
\end{aligned}
$$

Hence

$$y_3 = y_4 = q\alpha_1^{q+1}$$

and

$$q\alpha_3^{q+1} = y_3 + y_4 = 2q\alpha_1^{q+1}.$$

Showing that

$$\alpha_3 = \sqrt[(q+1)]{2}\alpha_1.$$

Since $y_3 > 0$, it should hold that $\frac{1}{\alpha_1} + \frac{1}{\alpha_3} = 25$. This gives $\alpha_1 = \frac{1+\frac{1}{\sqrt[(q+1)]{2}}}{25}$. It can now be readily verified that the other KKT conditions also hold.

Note that the solution in the example is unique since the cost function is strictly convex. Hence, the solution depends on the power function and also on the $(q + 1)$th-root of the number of producing tasks (here 2). This makes it impossible to create an algorithm that finds a single minimiser for a wide range of power functions, as was done in section 3.2. Furthermore, the dependencies between the tasks also influence the optimal solution. In the example, it can be beneficial to increase the speed of the consumer, such that the speed of multiple producers can be decreased. This also explains why the solution depends on the number of producers and the power function. If there another task is added that has to wait for one of the producers, it becomes harder to find the solution analytically. In case of many tasks depending on each other, it is hard to find a solution analytically. Improving energy efficiency of multi processor systems is an important, but hard problem and has to be considered for future research.

Chapter 7

Conclusions

Although the energy minimisation problem for real-time systems with N tasks is an infinite dimensional problem, it can be reduced to an N-dimensional convex problem. For the case that the power function p is convex and the energy per work $\bar{p}$ is non-decreasing and convex, an algorithm is given that gives an analytic solution that does not depend on p. In the optimal solution given by this algorithm, the scaling factors are non-increasing. In case the problem is strictly convex, the solution is unique and the scaling factors have to be non-increasing. When the scaling factors have to be chosen from a finite set A, the result of the previously discussed algorithm can be rewritten to an optimal solution that only uses scaling factors that are in A. It was shown that many existing algorithms from the literature can only be applied when $p(0) = 0$, while the algorithm given in this thesis does not have this restriction. This makes the algorithm presented in this thesis a significant contribution to what can be found in the literature. Especially in the (realistic) case that static power is present.

In practice, it turns out that the speed is only changed rarely when offline optimisation is used. For instance, for the playback of a specific 30 minute video sequence, the speed was changed seven times. When only a finite set of speeds (of size M), the minimum number of changes of the speed is bounded from above by $M - 1$. This makes offline optimisation a technique that might even be usable when the costs for changing the speed are relatively high, since changing the speed rarely occurs.

The online optimisation problem depends on predictions of future work. In case the predictions are perfect, the online energy minimisation approach presented in this thesis gives very good results. The energy consumption is almost as low as with offline energy minimisation.

Appendix **A**

Karush-Kuhn-Tucker conditions

This section explains how certain continuous non-linear minimisation problems can be solved. As an illustration of the kind of problems that can be solved, an example is be given.

Example A.1 Consider the following minimisation problem

$$\begin{aligned} \min_{x_1,x_2} \quad & x_1^2 + 2x_1 + x_2^2 + \frac{1}{2}x_1x_2 + \frac{1}{2}x_2 \\ \text{s.t.} \quad & -x_1 - x_2 + c \leq 0. \end{aligned}$$

To find the minimum, one can take the minima of $f(x_1, x_2) = x_1^2 + 2x_1 + x_2^2 + \frac{1}{2}x_1x_2 + \frac{1}{2}x_2$ and the minima of f on the boundaries of the set defined by $g(x_1, x_2) = -x_1 - x_2 + c \leq 0$. The lowest value gives the solution of the minimisation problem. First the critical value $\nabla f(x_1, x_2) = 0$ is located:

$$\nabla f(x_1, x_2) = \begin{bmatrix} 2x_1 + 2 + \frac{1}{2}x_2 \\ 2x_2 + \frac{1}{2} + \frac{1}{2}x_1. \end{bmatrix} = 0$$

These equations have the unique solution $x_1 = -1, x_2 = 0$. Since the Hessian of f is positive definite, this is a local minimiser of f. This solution is in the feasible set, if $-x_1 - x_2 + c \leq 0$, hence $c \leq -1$. To find the minima on the boundary of the feasible set, parametrised by $g(x) = -x_1 - x_2 + c$, Lagrange multipliers can be used. Now solve

$$\begin{aligned} 0 &= 2x_1 + 2 + \frac{1}{2}x_2 - \lambda \\ 0 &= 2x_2 + \frac{1}{2} + \frac{1}{2}x_1 - \lambda \\ 0 &= -x_1 - x_2 + c. \end{aligned}$$

Solving these equations give $x_1 = \frac{c-1}{2}$ and $x_2 = \frac{c+1}{2}$. It turns out that this is a minimiser and this is a minimum of the constrained minimisation problem when $c > -1$ because then the global minimiser of f is not in the feasible set.

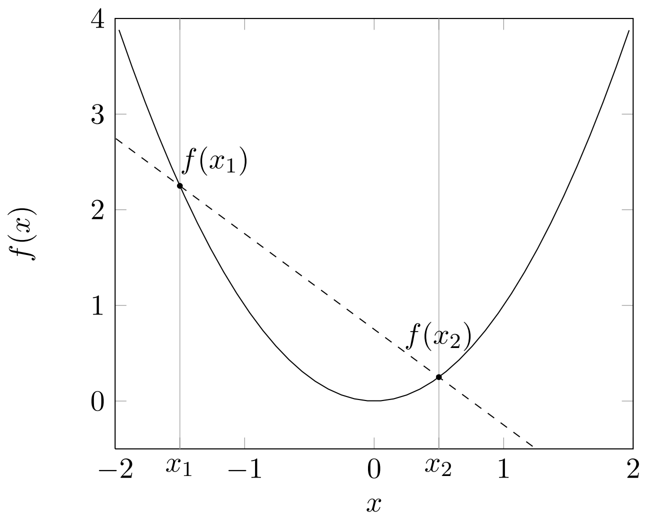


Figure A.1: Convex function

When an optimisation problem is strictly convex, as will be discussed in the next sections, every local minimiser is a global minimiser. Such problems have certain structure that can be exploited to find a solution. In this section the Karush-Kuhn-Tucker conditions are introduced. If these conditions are satisfied, a global minimum is found.

First some theory on convex functions is treated, enough to make this thesis self-contained. After a basic introduction into the theory of convex functions, an introduction into convex optimisation is given. Most definitions from this appendix are copied from, or based on, definitions in [4].

A.1 Convex functions

Definition A.1 (Convex set) The set $\mathcal{C} \subset \mathbb{R}^n$ is convex, if for all $x_1 \in \mathcal{C}$, $x_2 \in \mathcal{C}$ and $0 \leq \lambda \leq 1$ also $\lambda x_1 + (1-\lambda)x_2 \in \mathcal{C}$. □

Definition A.2 (Convex function) Given is a convex set $\mathcal{C} \subset \mathbb{R}^n$. A function $f : \mathcal{C} \to \mathbb{R}$ is convex, if for all $x_1 \in \mathcal{C}$, $x_2 \in \mathcal{C}$ and $0 \leq \lambda \leq 1$:

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda)f(x_2). \tag{A.1}$$

□

Definition A.3 (Strictly convex function) Given is a convex set $\mathcal{C} \subset \mathbb{R}^n$. A convex function $f : \mathcal{C} \to \mathbb{R}$ is strictly convex, if for all $x_1 \in \mathcal{C}$, $x_2 \in \mathcal{C}$ (with $x_1 \neq x_2$) and $0 < \lambda < 1$:

$$f(\lambda x_1 + (1-\lambda)x_2) < \lambda f(x_1) + (1-\lambda)f(x_2). \tag{A.2}$$

□

Examples of convex functions are e^x, x^2 and any norm $||x||$. The (strictly) convex function $f(x) = x^2$ is shown in figure A.1. Since $f(x)$ is strictly convex, the function, on $[x_1, x_2]$ is below the line through $f(x_1)$ and $f(x_2)$, this is illustrated using the dashed line in fig. A.1.

Using the definition of a convex function, it can be proven $f(x) = x^2$ is strictly convex.

Example A.2 Consider $f(x) : \mathbb{R} \to \mathbb{R}$, then for all $x_1, x_2 \in \mathcal{C}$, $x_1 \neq x_2$ and for all $0 < \lambda < 1$:

$$\begin{aligned}
f(\lambda x_1 + (1-\lambda)x_2) &= (\lambda x_1 + (1-\lambda)x_2)^2 \\
&= \lambda^2 x_1^2 + 2(1-\lambda)\lambda x_1 x_2 + (1-\lambda)^2 x_2^2 \\
&= \lambda x_1^2 + (1-\lambda)x_2^2 + (\lambda^2 - \lambda)x_1^2 + ((1-\lambda)^2 - (1-\lambda))x_2^2 + 2\lambda(1-\lambda)x_1 x_2 \\
&= \lambda x_1^2 + (1-\lambda)x_2^2 + (\lambda^2 - \lambda)x_1^2 + (\lambda^2 - \lambda)x_2^2 - 2(\lambda^2 - \lambda)x_1 x_2 \\
&= \lambda x_1^2 + (1-\lambda)x_2^2 + (\lambda^2 - \lambda)(x_1^2 + 2x_1 x_2 + x_2^2) \\
&= \lambda x_1^2 + (1-\lambda)x_2^2 + (\lambda^2 - \lambda)(x_1 + x_2)^2 \\
&\leq \lambda x_1^2 + (1-\lambda)x_2^2.
\end{aligned}$$

The inequality is due to $\lambda^2 - \lambda \leq 0$. Hence $f(x) = x^2$ is convex. When $x_1 \neq x_2$ and $0 < \lambda < 1$, the inequality can be changes into a strict inequality, showing $f(x) = x^2$ is also a strictly convex function.

The following lemma is often used to proof many theorems related to convex functions.

Lemma A.1 For any convex function $f : \mathbb{R} \to \mathbb{R}$ and $x_1 < x_2 < x_3 < x_4$:

$$\frac{f(x_4) - f(x_3)}{x_4 - x_3} \geq \frac{f(x_2) - f(x_1)}{x_2 - x_1}.$$

Proof Omitted. ■

An important inequality is Jensen's inequality. This inequality is available in many forms. The finite form is given in the following theorem.

Theorem A.1 (Jensen's inequality (finite form)) Let f be a convex function on a convex set $\mathcal{C} \subset \mathbb{R}^n$. Given are the points $x_1, \ldots, x_m \in \mathcal{C}$ and $\lambda_1, \ldots, \lambda_m \geq 0$ with $\sum_{i=1}^m \lambda_i = 1$. Then

$$f\left(\sum_{i=1}^m \lambda_i x_i\right) \leq \sum_{i=1}^m \lambda_i f(x_i).$$

Proof This result can be proven using the definition of a convex function with induction on m. This proof can be found in [4]. ■

Theorem A.2 (Jensen's inequality (infinite form)) Let $f : [a, b] \to \mathbb{R}$ be a convex function. If $g : [0, 1] \to [a, b]$ is an integrable function and if $f \circ g$ is integrable, then:

$$f\left(\int_0^1 g(t)dt\right) \leq \int_0^1 f(g(t))dt.$$

Equality holds if and only if $g(t)$ is a constant function or if f is linear on $g([0, 1])$.

Proof The proof can be found in [16]. ■

Several properties of convex functions are used in this thesis. An important property is that the weighted sum of convex functions is convex.

Lemma A.2 Let $f_1, \ldots, f_M$ be (strictly) convex functions, $g(x) = \sum_{i=1}^{M} a_i f_i(x)$ and $\forall i : a_i > 0$. Then the function $g(x)$ is also (strictly) convex.

Proof

$$\begin{aligned} g(\lambda x_1 + (1-\lambda)x_2) &= \sum_{i=1}^{N} a_i f_i(\lambda x_1 + (1-\lambda)x_2) \\ &\leq \sum_{i=1}^{N} a_i \left[\lambda f_i(x_1) + (1-\lambda) f_i(x_2)\right] \\ &= \lambda \sum_{i=1}^{N} \left[a_i f_i(x_1)\right] + (1-\lambda) \sum_{i=1}^{N} \left[a_i f_i(x_2)\right] \quad = \lambda g(x_1) + (1-\lambda) g(x_2). \end{aligned}$$

Where the inequality becomes a strict inequality for strictly convex functions. ■

Lemma A.3 For a differentiable function $f : \mathcal{C} \to \mathbb{R}$ where $\mathcal{C}$ is a convex set
f is convex if and only if for all $\bar{x}, x \in \mathcal{C}$:

$$f(\bar{x}) - f(x) \leq \nabla f(\bar{x})^T (\bar{x} - x) \tag{A.3}$$

Proof

$$\begin{aligned} & f(\lambda x + (1-\lambda)\bar{x}) \leq \lambda f(x) + (1-\lambda) f(\bar{x}) \\ \Rightarrow & f(\lambda x + (1-\lambda)\bar{x}) - f(\bar{x}) \leq \lambda (f(x) - f(\bar{x}_2)) \\ \Rightarrow & \frac{f(\bar{x} + \lambda(x - \bar{x})) - f(\bar{x})}{\lambda} \leq f(x) - f(\bar{x}) \\ \Rightarrow & f(\bar{x}) - f(x) \leq -\frac{f(\bar{x} + \lambda(x - \bar{x})) - f(\bar{x})}{\lambda} \\ \Rightarrow & \lim_{\lambda \to 0} f(\bar{x}) - f(x) \leq -\lim_{\lambda \to 0} \frac{f(\bar{x} + \lambda(x - \bar{x})) - f(\bar{x})}{\lambda} \\ \Rightarrow & f(\bar{x}) - f(x) \leq \nabla f(\bar{x})^T (\bar{x} - x) \end{aligned}$$

and for all $x_1, x_2, \bar{x} \in \mathcal{C}$:

$$\begin{aligned}
&f(\bar{x}) - f(x) \leq \nabla f(\bar{x})^T(\bar{x} - x), \text{for all } \bar{x}, x \in \mathcal{C} \\
\Rightarrow &\lambda f(\bar{x}) - \lambda f(x_1) + (1-\lambda) f(\bar{x}) - (1-\lambda) f(x_2) \leq \lambda \nabla(\bar{x})^T(\bar{x} - x_1) + (1-\lambda)\nabla f(\bar{x})^T(\bar{x} - x) \\
\Rightarrow &f(\bar{x}) \leq \lambda f(x_1) + (1-\lambda) f(x_2) + \nabla f(\bar{x})^T(\lambda\bar{x} - \lambda x_1 + (1-\lambda)\bar{x} - (1-\lambda)x_2) \\
\Rightarrow &f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2).
\end{aligned}$$
■

There is also a second order condition for convexity, given in the following lemma.

Lemma A.4 Given is a convex set $\mathcal{C} \subseteq \mathbb{R}^n$. If for all $x \in \mathcal{C}$ holds that $\nabla^2 f(x) \leq 0$, $f(x)$ is convex. If for all $x \in \mathcal{C}$ holds that $\nabla^2 f(x) < 0$, $f(x)$ is strictly convex.

Proof Using second order Taylor's formula (see [5]), for some $\tau \in (0,1)$:

$$\begin{aligned}
f(x) &= f(\bar{x}) + \nabla f(\bar{x})^T(x - \bar{x}) + \frac{1}{2}(x-\bar{x})^T \nabla^2 f(\bar{x} + \tau(x - \bar{x}))(x - \bar{x}) \\
&\leq f(\bar{x}) + \nabla f(\bar{x})^T(x - \bar{x}),
\end{aligned}$$

now it follows that

$$f(x) - f(\bar{x}) \leq \nabla f(\bar{x})^T(x - \bar{x}).$$

If the Hessian is positive definite, the inequalities change into strict inequalities. From Lemma A.3, it directly follows that f is a (strictly) convex function. ■

Lemma A.4 is often used to proof convexity.

Example A.3 The functions $f : \mathbb{R} \to \mathbb{R}$ given by $f(x) = x^2$ and $g : \mathbb{R}^+ \to \mathbb{R}$ given by $g(x) = \frac{1}{x}$ are strictly convex. The second derivatives of these functions are respectively $f''(x) = 2 > 0$ and $g''(x) = \frac{2}{x^3} > 0$ or their domain, hence they are convex.

Lemma A.5 Given are a convex set $\mathcal{C} \subseteq \mathbb{R}^n$ and a convex function $f : \mathcal{C} \to \mathbb{R}$, which is twice differentiable. Then, $\nabla^2 f(x) \leq 0$.

Proof For all $x_1, x_2 \in \mathcal{C}$:

$$\begin{aligned}
&f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2) \\
\Rightarrow &\nabla f(\lambda x_1 + (1-\lambda)x_2)^T x_1 \leq f(x_1) - f(x_2) \\
\Rightarrow &x_1^T \nabla^2 f(\lambda x_1 + (1-\lambda)x_2)^T x_1 \leq 0 \\
\Rightarrow &x_1^T \nabla^2 f(x_2)^T x_1 \leq 0.
\end{aligned}$$
■

It should be noted that when $f(x)$ is strictly convex, it does not imply that the Hessian of f is positive definite. Take for instance $f(x) = x^4$, then $f''(x) = 12x^2$. However, $f''(0) = 0$, while $f(x)$ is strictly convex.

Lemma A.6 Let $\mathcal{C} \subseteq \mathbb{R}^n$ be a convex set and $f_1, \dots, f_n$ strictly convex functions from $\mathbb{R}$ to $\mathbb{R}$. The function

$$g(x) = \sum_{i=1}^{n} f_i(x_i)$$

is strictly convex.

Proof For all $x, y \in \mathbb{R}^n$, $x \neq y$ where $x = (x_1, \dots, x_n)^T$ and $y = (y_1, \dots, y_n)^T$:

$$\begin{aligned} g(\lambda x + (1-\lambda)y) &= \sum_{i=1}^{n} f_i(\lambda x_i + (1-\lambda)y_i) \\ &< \sum_{i=1}^{n} \lambda f_i(x_i) + (1-\lambda)f(y_i) \\ &= \lambda \sum_{i=1}^{n} [f_i(x_i)] + (1-\lambda)\sum_{i=1}^{n} [f_i(y_i)] \\ &= \lambda g(x) + (1-\lambda)g(y). \end{aligned}$$

Since x and y are different in at least one component, the strict inequality can always be used. ■

Example A.4 Take $x = (x_1, \dots, x_n)^T \in (0,1]^n$, $a_i > 0$ and

$$f(x) = \sum_{i=1}^{n} \frac{a_i}{x_i}.$$

This function is strictly convex, since $\frac{a_i}{x_i}$ is strictly convex on $(0,1]$ and $f(x)$ is a separable function in the form given by Lemma A.6. By Lemma A.6, it immediately follows that $f(x)$ is strictly convex.

A nice property of convex functions on a convex set $\mathcal{C}$, is that these functions are continuous on a subset (called the relative interior) of $\mathcal{C}$.

Definition A.4 (Relative interior) Let $\mathcal{C} \subseteq \mathbb{R}^n$ be a convex set. The point $x \in \mathcal{C}$ is in the relative interior of $\mathcal{C}$ if for all $\bar{x} \in \mathcal{C}$ there exists $\tilde{x} \in \mathcal{C}$ and $0 < \lambda < 1$ such that $x = \lambda\bar{x} + (1-\lambda)\bar{x}$. The set of relative interior points of the set $\mathcal{C}$ will be denoted by $\mathcal{C}^0$. □

Lemma A.7 Let $\mathcal{C}$ be a convex set and $f : \mathcal{C} \to \mathbb{R}$ be a convex function. Then f is continuous on the relative interior of $\mathcal{C}^0$ of $\mathcal{C}$.

PROOF This proof can be found in [4]. ■

A.2 Convex Optimisation

In convex optimisation one tries to find a global minimum of a convex function $f : \mathcal{C} \to \mathbb{R}$ where $\mathcal{C}$ is a convex set. A nice property of this problem is that every local minimum is a global minimum as the following lemma shows.

Lemma A.8 Given the convex set $\mathcal{C} \subseteq \mathbb{R}^n$ and the convex function $f : \mathcal{C} \to \mathbb{R}$. Every local minimum of f is also a global minimum.

PROOF If $\mathcal{C} = \emptyset$, it is trivially true. Now it will be proven for $\mathcal{C} \neq \emptyset$. Assume the statement is not true and there exists a local (but not global) minimum $\bar{x} \in \mathcal{C}$ and a $x^* \in \mathcal{C}$, for which $f(x^*) < f(\bar{x})$. Since $\bar{x}$ is a local minimum, there is an ϵ such that

$$||\bar{x} - x|| < \epsilon \Rightarrow f(\bar{x}) \leq f(x).$$

Now define

$$y = \lambda x^* + (1 - \lambda)\bar{x}$$

with a λ such that

$$0 < \lambda < \min\left(\frac{\epsilon}{2||\bar{x} - x^*||}, 1\right).$$

Since $\mathcal{C}$ is convex and $0 \leq \lambda \leq 1$, $y \in \mathcal{C}$. Then

$$\begin{aligned} ||\bar{x} - y|| &= ||\bar{x} - \lambda x^* - \bar{x} + \lambda\bar{x}|| \\ &= ||\lambda(\bar{x} - x^*)|| \\ &= \lambda||\bar{x} - x^*|| \\ &< \frac{\epsilon}{2||\bar{x} - x||}||\bar{x} - x^*|| \\ &= \frac{\epsilon}{2}. \end{aligned}$$

And then $f(\bar{x}) \leq f(y)$ and

$$\begin{aligned} f(y) &= f(\lambda x^* + (1 - \lambda)\bar{x}) \\ &\leq \lambda f(x^*) + (1 - \lambda)f(\bar{x}) \\ &< \lambda f(\bar{x}) + (1 - \lambda)f(\bar{x}) \\ &= f(\bar{x}). \end{aligned}$$

Now $\bar{x}$ is not a local minimum, which leads to a contradiction. ■

If the function f is strictly convex, an even stronger result can be given.

Lemma A.9 Given the convex set $\mathcal{C}$ and the strictly convex function $f : \mathcal{C} \to \mathbb{R}$. If there is a global minimum, it is unique.

PROOF It was shown in Lemma A.8 that the minima are all global minima. Assume the statement is false and there are two global minimisers x_1 and x_2. Then

$$\begin{aligned} f(\lambda x_1 + (1-\lambda)x_2) &< \lambda f(x_1) + (1-\lambda) f(x_2) \\ &= \lambda f(x_1) + (1-\lambda) f(x_1) \\ &= f(x_1). \end{aligned}$$

This shows that x_1 and x_2 are not global minima, which leads to a contradiction. ■

Often a minimum of a convex function f over a convex set $\mathcal{C}$ is desired. For many problems, it is sufficient to assume that the set $\mathcal{C}$ is described using inequality constraints.

Definition A.5 (Convex optimisation problem) The convex optimisation problem is defined as

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} f(x) \\ &\text{s.t. } g_j(x) \leq 0 \qquad , j \in \{1, \dots, m\}, \quad \text{(CO)} \\ &\qquad h_j(x) = 0 \qquad , j \in \{1, \dots, l\}, \end{aligned}$$

where $g_1, \dots, g_m$ (the inequality constraints), $h_1, \dots, h_l$ (the equality constraints) and f (the objective function or cost function) are convex functions on $\mathbb{R}^n$. □

In the remainder of this section, it is assumed (CO) has no equality constraints ($l = 0$), section A.3 will discuss equality constraints.

Definition A.6 (Active constraint at $\bar{x}$) A constraint g_i is active at $\bar{x}$ if $g_i(\bar{x}) = 0$. The index set of all active constraints at $\bar{x}$ is denoted by $I_{\bar{x}}$. □

Definition A.7 (The feasible set $\mathcal{F}$) The feasible set, or set of feasible points, $\mathcal{F}$, is defined as

$$\mathcal{F} = \{x \in \mathbb{R}^n | g_j(x) \leq 0, j = 1, \dots, m\},$$

where $g_1, \dots, g_m$ are convex functions on $\mathbb{R}^n$. □

The values in $\mathcal{F}$ for which the minimum is attained will be called solutions. The set $\mathcal{F}$ is convex, which is shown by the following lemma.

Lemma A.10 The feasible set $\mathcal{F}$ is a convex set.

PROOF For all $x, y \in \mathcal{F}$ and $0 \leq \lambda \leq 1$ it has to be proven that $z = \lambda x + (1-\lambda) y \in \mathcal{F}$. This is the case if and only if for all i: $g_i(z) \leq 0$ and then:

$$\begin{aligned} g_i(z) &= g_i(\lambda x + (1-\lambda) y) \\ &\leq \lambda g_i(x) + (1-\lambda) g_i(y) \\ &\leq 0. \end{aligned}$$

■

The next theorem gives the sufficient conditions for optimality.

Theorem A.3 (Karush-Kuhn-Tucker (KKT) Optimality Condition) The value $\bar{x} \in \mathbb{R}^n$ is a solution of (CO) if there exists a $\bar{y} \geq 0$ such that the following equations hold

(i) $g_j(\bar{x}) \leq 0, \text{for all } j \in \{1, \dots, m\}$,

(ii) $0 = \nabla f(\bar{x}) + \sum_{j=1}^{m} \bar{y}_j \nabla g_j(\bar{x})$,

(iii) $\sum_{j=1}^{m} \bar{y}_j g_j(\bar{x}) = 0$,

(iv) $\bar{y} \geq 0$.

Proof This proof is due to [6].

$$\begin{aligned}
f(x) - f(\bar{x}) &\geq (\bar{x} - x)^T \nabla f(\bar{x}) && \text{(A.4)}\\
&= -(\bar{x} - x)^T \textstyle\sum_{i=1}^{m} \bar{y}_i \nabla g_i(\bar{x}) && \text{(A.5)}\\
&\geq -\left[\textstyle\sum_{i=1}^{m} \bar{y}_i (g_i(x) - g_i(\bar{x}))\right] && \text{(A.6)}\\
&= -\left[\textstyle\sum_{i=1}^{m} \bar{y}_i g_i(x)\right] && \text{(A.7)}\\
&\geq 0 && \text{(A.8)}
\end{aligned}$$

For eq. (A.4), the lemma A.3 is used. The KKT condition (ii) is used for eq. (A.5). The KKT condition $\bar{y} \geq 0$ and lemma A.3 is used in eq. (A.6). For eq. (A.7), KKT condition (iii) is used. In the last step, it is used that $\bar{y} \geq 0$ and $g(\bar{x}) \leq 0$. ■

In this theorem, the variables $\bar{y}$ were introduced. These variables are called the *dual variables.*

Figure A.2 offers some insights in the KKT conditions. The feasible region is defined by the constraints g_1, g_2 and g_3. The constraint g_i is active when $g_i(x) = 0$, the boundaries of the feasible region are defined by these active constraints. In the direction $-\nabla g_i$ from $g_i(x) = 0$, the constraint is satisfied. The direction of the negative gradient of f is indicated. This shows that as long as the boundaries or a minimum are not reached, one can move along the gradient towards a minimum (if exists).

Example A.5 Consider the following convex minimisation problem

$$\begin{aligned}
\min_{x_1, x_2} \quad & x_1^2 + 2x_1 + x_2^2 + \frac{1}{2}x_1x_2 + \frac{1}{2}x_2\\
\text{s.t.} \quad & -x_1 - x_2 + c \leq 0.
\end{aligned}$$

This is the problem from Example A.1 and will now be solved using the KKT conditions.

The gradients of the function $f(x_1, x_2) = x_1^2 + 2x_1 + x_2^2 + \frac{1}{2}x_1x_2 + \frac{1}{2}x_2$ and $g(x_1, x_2) = -x_1 - x_2 + c$ are required. These are

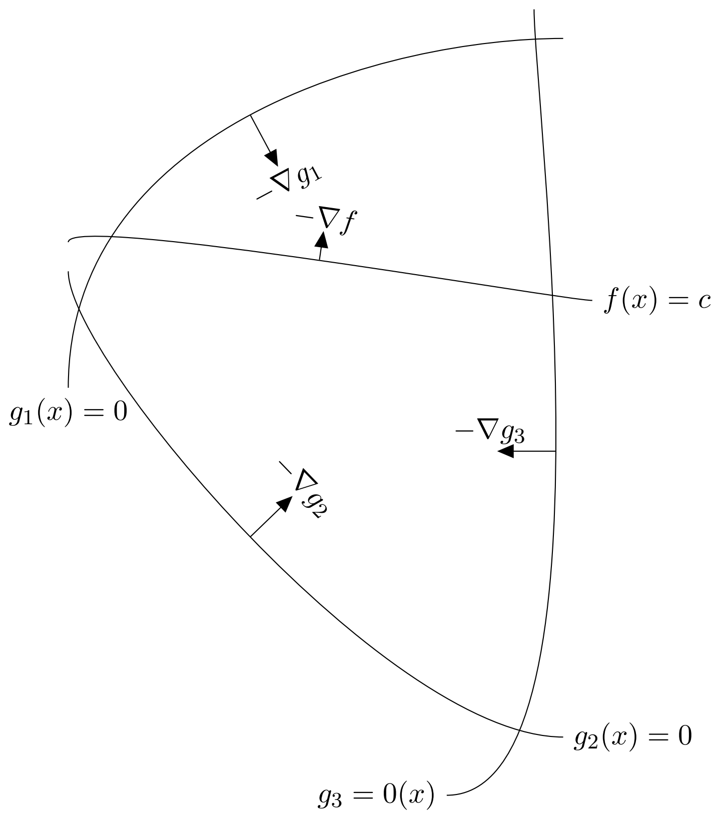


Figure A.2: KKT conditions

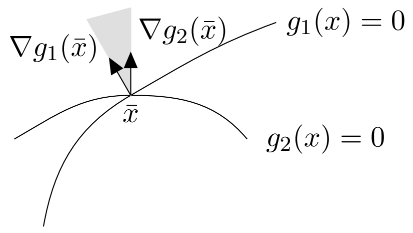


Figure A.3: KKT conditions at active constraints

$$\nabla f(x_1, x_2) = \begin{bmatrix} 2x_1 + 2 + \frac{1}{2}x_2 \\ 2x_2 + \frac{1}{2} + \frac{1}{2}x_1 \end{bmatrix}$$

and

$$\nabla g(x_1, x_2) = \begin{bmatrix} -1 \\ -1. \end{bmatrix}$$

The following equations should now hold

$$0 \geq -x_1 - x_2 + c \tag{A.9}$$
$$0 = 2x_1 + 2 + \frac{1}{2}x_2 - y \tag{A.10}$$
$$0 = 2x_2 + \frac{1}{2} + \frac{1}{2}x_1 - y \tag{A.11}$$
$$0 = y(-x_1 - x_2 + c) \tag{A.12}$$
$$y \geq 0. \tag{A.13}$$

From eq. (A.10) and eq. (A.11) follows

$$2x_1 + 2 + \frac{1}{2}x_2 = 2x_2 + \frac{1}{2} + \frac{1}{2}x_1 \Leftrightarrow x_1 + 1 = x_2.$$

Now eq. (A.13) give two possibilities:

- $y = 0$

  The equations

  $$\begin{cases} 2x_1 + 2 + \frac{1}{2}x_2 = 0 \\ 2x_2 + \frac{1}{2} + \frac{1}{2}x_1 \end{cases}$$

  have the solution $x_1 = -1$, $x_2 = 0$. Furthermore $x_1 + x_2 \geq c \Rightarrow c \leq -1$.

- $y > 0$

  Because of eq. (A.12), $-x_1 - x_2 + c = 0$, then by using $x_1 + 1 = x_2$

  $$x_1 = \frac{c-1}{2}$$
  $$x_2 = \frac{c+1}{2}$$

  Furthermore

$$\begin{aligned} y &= 2x_1 + 2 + \frac{1}{2}x_2 \\ &= 2x_1 + 2 + \frac{1}{2}x_1 + \frac{1}{2} \\ &= \frac{5}{2}x_1 + \frac{5}{2} \\ &= \frac{5c}{4} - \frac{5}{4} + \frac{5}{2} \\ &= \frac{5c}{4} + \frac{5}{4} \\ &> 0 \end{aligned}$$

hence, $c > -1$.

When $y = 0$, the constraint is active and the solution is found on the border of the feasible set. Since the border depends on c, the solution depends on c. Then $y > 0$, the constraint is in the interior of the feasible set where a single solution can be found which is the global minimiser of the unconstrained problem. This is related to how Example A.1 was solved.

If some additional conditions hold, the conditions from theorem A.3 are necessary and sufficient. One such condition is the Slater condition, which is defined as follows.

Definition A.8 (Slater condition) The point $\bar{x}$ satisfies the Slater condition if

$$\begin{aligned} &g_j(\bar{x}) < 0 && \text{for all } j \text{ where } g_j \text{ is nonlinear,} \\ &g_j(\bar{x}) \leq 0 && \text{for all } j \text{ where } g_j \text{ is linear,} \end{aligned}$$

this point $\bar{x}$ is called a Slater point. □

Definition A.9 (Slater regular) The convex optimisation problem (CO) is called Slater regular when a Slater point exists. □

Theorem A.4 Assume that (CO) is Slater regular. Then the KKT conditions are necessary and sufficient for optimality.

Proof Omitted. The proof can be found in [4], pages 36–46. ■

It is important to note that not every solution of (CO) has to satisfy the KKT conditions, it is only necessary when the (CO) is Slater regular.

Example A.6 Consider

$$\min_{x \in \mathbb{R}} \quad x \tag{A.14}$$

$$\text{s.t.} \quad x^2 \leq 0. \tag{A.15}$$

The feasible set $\mathcal{F} = \{0\}$ consists of one point, 0, where the Slater conditions do not hold. Therefore the problem is not Slater regular. Since $\nabla f = 1$ and $\nabla g = 2x$, there is no $y > 0$ such that $1 = -2xy$. Because of this, the KKT conditions do not hold for 0, while this is the solution op the optimisation problem.

A.3 Equality constraints

Again consider (CO):

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & g_j(x) \leq 0 && \text{for all } j \in \{1, \dots, m\}, \quad \text{(CO)} \\ & h_j(x) \leq 0 && \text{for all } j \in \{1, \dots, l\}. \end{aligned}$$

In the previous section it was assumed $l = 0$, hence no equality constraints. However, equality constraints can be rewritten to inequality constraints. Take an equality constraint $h_j(x) \leq 0$, this can be written as $g_{m+2j}(x) \leq 0$ and $-g_{m+2j+1}(x) \leq 0$. If h_j is convex, $-h_j$ (and then also $-g_{m+2j+1}$) is also convex when h_j is linear. In the KKT conditions, the constraint occurs twice, for instance $\cdots + y_{m+2j}\nabla h_j + y_{m+2j}(-\nabla h_j)$. Now the constraints can be replaced by a single multiplier λ_j. The KKT conditions can now be written as

(i) $g_j(\bar{x}) \leq 0, \text{for all } j \in \{1, \dots, m\}$,

(ii) $h_j(\bar{x}) = 0, \text{for all } j \in \{1, \dots, l\}$,

(iii) $0 = \nabla f(\bar{x}) + \sum_{j=1}^{m} \bar{y}_j \nabla g_j(\bar{x}) + \sum_{j=1}^{l} \lambda_j \nabla h_j(\bar{x})$,

(iv) $\sum_{j=1}^{m} \bar{y}_j g_j(\bar{x}) + \sum_{j=1}^{l} \lambda_j g_j(\bar{x}) = 0$,

(v) $\bar{y}_j \geq 0, \text{for all } j \in \{1, \dots, m\}$.

Bibliography

[1] E. Akyol and M. van der Schaar. Complexity model based proactive dynamic voltage scaling for video decoding systems. *IEEE Transactions on Multimedia*, 9(7):1475, 2007.

[2] A. Alimonda, S. Carta, A. Acquaviva, A. Pisano, and L. Benini. A feedback-based approach to DVFS in data-flow applications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(11):1691–1704, 2009.

[3] S. H. Chan, D. Vo, and T. Q. Nguyen. Subpixel motion estimation without interpolation. http://videoprocessing.ucsd.edu/ stanleychan/research/MotionEstimation.html.

[4] E. de Klerk, C. Roos, and T. Terlaky. Nonlinear optimization. Lecture notes, University of Waterloo, February 2006.

[5] U. Faigle, W. Kern, and G. Still. *Algorithmic Principles of Mathematical Programming (Texts in the Mathematical Sciences)*. Springer, 2010.

[6] M. A. Hanson. On sufficiency of the kuhn-tucker conditions. *Journal of Mathematical Analysis and Applications*, 80(2):545–550, 1981.

[7] C.-h. Hsu and W.-c. Feng. When discreteness meets continuity: Energy-optimal DVS scheduling revisited. Technical Report LA-UR 05-3104, Los Alamos National Laboratory, February 2005.

[8] W. Huang and Y. Wang. An optimal speed control scheme supported by media servers for low-power multimedia applications. *Multimedia Systems*, 15(2):113–124, 2009.

[9] W.-C. Kwon and T. Kim. Optimal voltage allocation techniques for dynamically variable voltage processors. *ACM Transactions on Embedded Computing Systems*, 4(1):211–230, 2005.

[10] X. Liu, P. Shenoy, and W. Gong. A time series-based approach for power management in mobile processors and disks. In *Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video - NOSSDAV '04*, pages 74–79, 2004.

[11] Z. Lu, J. Hein, M. Humphrey, M. Stan, J. Lach, and K. Skadron. Control-theoretic dynamic frequency and voltage scaling for multimedia workloads. In *Proceedings of the international conference on Compilers, architecture, and synthesis for embedded systems - CASES '02*, pages 156–163, 2002.

[12] Z. Lu, J. Lach, M. Stan, and K. Skadron. Reducing multimedia decode power using feedback control. In *Proceedings 21st International Conference on Computer Design*, pages 489–496, 2003.

[13] A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony, and R. Rajkumar. Critical power slope: understanding the runtime effects of frequency scaling. In *Proceedings of the 16th international conference on Supercomputing*, ICS '02, pages 35–44, New York, NY, USA, 2002. ACM.

[14] J. A. Pouwelse, K. G. Langendoen, R. L. Lagendijk, and H. J. Sips. Power-aware video decoding. In *22nd Picture Coding Symposium, Seoul, Korea*, pages 303–306, Seoul, Korea, April 2001. Citeseer.

[15] M. K. Steliaros. Block-matching motion compensation. http://www.dcs.warwick.ac.uk/research/mcg/bmmc/index.html.

[16] W. R. Wade. *Introduction to Analysis (3rd Edition)*. Prentice Hall, 2003.

[17] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark. Formal online methods for voltage/frequency control in multiple clock domain microprocessors. In *Proceedings of the 11th international conference on Architectural support for programming languages and operating systems - ASPLOS-XI*, pages 248–259, 2004.

[18] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 374–382, 1995.

[19] T. Zitterell and C. Scholl. A probabilistic and energy-efficient scheduling approach for online application in real-time systems. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 42–47, New York, NY, USA, 2010. ACM.