CHAPTER 1

Introduction

1.1 What Is an Online Algorithm

We begin the discussion on what an online algorithm is using a simple example. Consider a vector $\mathbf{X} = (x_1, x_2, \ldots, x_n)$, where $x_i \in \mathbb{R}^+$. Let the objective be to select that element of $\mathbf{X}$ that has the largest value $x_{i^\star}$, where $i^\star = \arg\max_i x_i$. In the usual setting, called **offline**, when the full vector $\mathbf{X}$ is observable/available, the best element $i^\star$ can be found trivially. Here trivially means that it is always possible to find $i^\star$, disregarding the complexity of finding it.

Next, consider an **online** setting, where at step t, an online algorithm observes x_t, the t^{th}-element of $\mathbf{X}$, and needs to make one of the following two decisions: (i) Either select x_t, and declare it to be of the largest value, in which case no future element of $\mathbf{X} \backslash \mathbf{X}^t$ is presented to it, where $\mathbf{X}^t = (x_1, x_2, \ldots, x_t)$, or (ii) does not select x_t, and moves on to observe x_{t+1}, but in this case, it cannot later select any of the elements of $\mathbf{X}^t$ seen already.

Thus, an online algorithm is limited in its view of the input and has to make decisions after observing partial inputs that cannot be changed in the future. Under these online/causal constraints, the online algorithm's objective is still to select the best element $i^\star$. Clearly, this is a challenging task, and depends on the order in which elements of $\mathbf{X}$ are presented, i.e., the online algorithm's decisions might be different with input $\mathbf{X}$ or $\pi(\mathbf{X})$, where π is any permutation, since the online algorithm makes decisions after observing partial inputs. In fact one can argue that no online algorithm can always select the best element $i^\star$ unlike an offline algorithm. Hence, a trivial problem in the offline setting turns out to be quite difficult in the online setting, where it is known as the **secretary** problem. We will discuss the secretary problem in detail in Chapter 7.

To better understand the definition of an online algorithm, consider another example. Let $\mathbf{Y} = \{y_1, y_2, \ldots, y_n\}$ be a set of elements, where y_i represents the 'size' of element i with $0 < y_i < 1, 1 \leq i \leq n$. The objective is to partition the n elements of $\mathbf{Y}$ into as few disjoint subsets as possible, such that the sum of the sizes of all items in each subset is at most 1 (capacity constraint), and the union of all subsets is equal to $\mathbf{Y}$. Disregarding the complexity of finding the solution, if the whole set $\mathbf{Y}$ is available (the offline setting), it is always possible to find such a partition, e.g., using complete enumeration. This problem is popularly called the **bin-packing** problem.

In the online setting, let $\mathbf{Y}^t = \{y_1, y_2, \ldots, y_t\}$ be the prefix of $\mathbf{Y}$ that is revealed until time step t. At each step t, with the knowledge of only $\mathbf{Y}^t$, an online algorithm has to compute its partition under the constraint that in the future, partition can only be augmented. That is, at step t, by observing the newest element t with size y_t, the algorithm can either assign y_t to any of the existing subsets (if the constraint on the sum of the sizes is satisfied) or open a new subset. It cannot 'disturb' the elements already assigned to subsets.

https://doi.org/10.1017/9781009349178.002 Published online by Cambridge University Press

Similar to the secretary problem, the number of subsets used by an online algorithm will depend on the order of arrival of elements of $\mathbf{Y}$, i.e., possibly different with $\mathbf{Y}$ or $\pi(\mathbf{Y})$, while the number of subsets used by the offline algorithm does not depend on π. Compared to the offline setting, complete enumeration is useless for finding an online algorithm given the limited information setting it has to work with in the presence of the augmentation requirement. Thus, even with unlimited complexity, finding an optimal online algorithm is non-trivial.

With these two examples under our belt, we next define an online algorithm abstractly. In general, let σ be the full input, σ^t its t-length prefix, and σ_t the input revealed at time t. A deterministic online algorithm $\mathcal{A}$, at step t, observes σ_t, has the knowledge of σ^{t-1}, and makes an irrevocable decision at time t denoted by D_t using only the information revealed so far, i.e., σ^t. The overall cost of $\mathcal{A}$ with input σ is then represented as

$$f_{\mathcal{A}}(\sigma) = f(D_1(\sigma^1), D_2(\sigma^2), \dots, D_{|\sigma|}(\sigma^{|\sigma|})),$$

where f is the objective function.

To highlight the challenge faced by an online algorithm, consider that the input σ is partitioned into two equal sized parts σ_1 and σ_2, i.e., $\sigma = (\sigma_1, \sigma_2)$. Moreover, let σ_2 take two possible values σ_{21} and σ_{22}. An online algorithm $\mathcal{A}$ until the end of sub-input σ_1 does not know whether σ_2 is equal to σ_{21} or σ_{22}. So the immediate question is: what should $\mathcal{A}$ do until the end of sub-input σ_1?

One possibility is that $\mathcal{A}$ assumes that σ_2 is equal to σ_{21} (σ_{22}) and makes its decisions appropriately until the end of sub-input σ_1. However, if σ_2 is equal to σ_{22} (σ_{21}), then in hindsight $\mathcal{A}$'s cost could turn out be too large compared to if it had guessed the correct input σ_{22} (σ_{21}). A more prudent alternative for $\mathcal{A}$ is to assume that an adversary is going to choose σ_2 among σ_{21} and σ_{22} depending on its decisions until the end of sub-input σ_1 to maximize $\mathcal{A}$'s overall cost. Consequently, $\mathcal{A}$ should make its decisions until the end of sub-input σ_1, such that in hindsight its cost is not too large in either of the two cases. This approach appears more reasonable for the online paradigm when no assumption about the future inputs is made.

Essentially, what this entails is that $\mathcal{A}$ should not overly commit to σ_2 being either σ_{21} or σ_{22}, and keep equal 'distance' from the two hindsight optimal algorithms corresponding to (σ_1, σ_{21}) and (σ_1, σ_{22}). As a result, the cost of $\mathcal{A}$ will be larger than the hindsight optimal algorithm for both cases (σ_1, σ_{21}) and (σ_1, σ_{22}), however, will be guarded against adversarial choice of σ_{21} or σ_{22}. The ensuing penalty compared to the hindsight optimal algorithm controls the performance of $\mathcal{A}$. We next formalize this idea to describe the performance metric for online algorithms.

Consider the offline setting, where the complete input σ is available at time 0 non-causally, and OPT is an optimal algorithm knowing the complete input σ. One could equivalently think of OPT as an hindsight optimal algorithm. For input σ, let the cost of OPT be $f_{\mathsf{OPT}}(\sigma)$, where note that OPT does not make sequential or causal decisions, unlike an online algorithm. Following on from the preceding discussion, since OPT is the hindsight optimal algorithm, we can benchmark the performance of an online algorithm $\mathcal{A}$ against the OPT under the uncertain future input model which could possibly be adversarial.

One such performance metric is the **regret**, that for an online algorithm $\mathcal{A}$ is defined as

$$\mathbf{R}(\mathcal{A}) = \max_{\sigma} |f_{\mathsf{OPT}}(\sigma) - f_{\mathcal{A}}(\sigma)|,$$

https://doi.org/10.1017/9781009349178.002 Published online by Cambridge University Press

the largest difference between the cost of the OPT and that of an online algorithm, over the worst case input. The maximization over the input σ represents the adversary that can choose input at any time depending on prior decisions made by $\mathcal{A}$ to reflect the maximum penalty $\mathcal{A}$ has to pay. With regret as the metric of choice, the quest is to find an online algorithm whose regret is sub-linear in the size $|\sigma|$ of the input σ so that the regret normalized with $|\sigma|$ goes to zero. It turns out that for most of the online problems of interest, sub-linear regret is not possible, and hence a weaker metric called the **competitive ratio**, defined next, is used far more popularly.

For a minimization problem (where the goal is to minimize an objective function), the **competitive ratio** of a deterministic online algorithm $\mathcal{A}$ is defined as

$$\mu_{\mathcal{A}} = \max_{\sigma} \frac{f_{\mathcal{A}}(\sigma)}{f_{\mathsf{OPT}}(\sigma)}, \tag{1.1}$$

i.e., the largest ratio of the cost of an online algorithm and the OPT, maximized over all possible inputs. For a maximization problem, the max is replaced by a min to define

$$\mu_{\mathcal{A}} = \min_{\sigma} \frac{f_{\mathcal{A}}(\sigma)}{f_{\mathsf{OPT}}(\sigma)}. \tag{1.2}$$

Similar to the regret definition, the maximization (minimization) over the input σ to define the competitive ratio represents the adversary that captures the maximum multiplicative penalty $\mathcal{A}$ pays compared to OPT.

The competitive ratio is an unforgiving metric, since an online algorithm that has its cost close to that of the OPT for all almost all inputs, except a few where it is comparatively very large, will be termed bad. To keep the competitive ratio small an online algorithm has to make sure that its cost is close to that of the OPT for all possible inputs. With competitive ratio as the performance metric, the goal is to find online algorithms with constant (and smallest/largest possible depending on whether it is a min/max problem) competitive ratios, i.e., independent of all parameters of the problem specification, and the input. Competitive ratio is a multiplicative metric as compared to the regret that is an additive metric, and thus provides a weaker guarantee even when it is a constant.

One may ask why compare an online algorithm with the OPT (offline setting), isn't that too unfair? The answer to that is multifold. First is the fact that OPT is the absolute benchmark, and if the competitive ratio of an online algorithm is small, this implies that the online algorithm is nearly making optimal decisions even with partial inputs. Second is the difficulty in even defining an optimal online algorithm under the uncertain future input model. Even though the competitive ratio is a pessimistic metric, for many of the problems of interest, the competitive ratio can be shown to be small constants, which is remarkable, and makes the comparison with OPT meaningful. Moreover, considering the competitive ratio metric, robustness to input perturbation is embedded in the online algorithm, since the competitive ratio guarantee is with respect to the worst case input. An added feature of most of the analysis on online algorithms is that their competitive ratios can be bounded without explicitly characterizing the OPT.[1]

[1] For most of the problems considered in the chapter, finding OPT that is efficient (polynomial time complexity) is not possible. Thus, the obvious step is to approximate the cost of OPT using efficient algorithms. [1] is an excellent reference for approximation algorithms.

https://doi.org/10.1017/9781009349178.002 Published online by Cambridge University Press

We will discuss the implications of the definition of competitive ratio in a little more detail in Section 1.3. We next outline some of the applications that motivate the study of online algorithms.

1.2 Why Study Online Algorithms

In this section, we discuss some of the motivating online settings that are driven by important practical problems.

Buy/Rent Problems Consider that you need an expensive resource, e.g., a luxury car or a yacht. As is natural, buying it outright costs you much more than renting it. If you knew for how long you were going to need that resource, the problem is easy to solve: pick the choice that costs less. The fact that makes the problem interesting is that you do not know how long you need that resource for, which could be on account of a variety of reasons, such as weather, future appointments, emergencies, etc. Thus, each day you have to decide whether to keep renting or buy, given the unknown remaining use-time. This is a canonical online problem, where the uncertainty is in the number of days for which you need the resource. This problem is popularly known as the **ski-rental** problem, named so for the usual unpredictability of the remaining length of the ski-season.

The ski-rental problem is also closely connected to problems with applications in different areas such as snoopy caching, where there are shared cached systems with a common memory, or the TCP acknowledgment problem, where TCP is the backbone protocol for sending packets reliably over the internet.

Memory Management in Operating Systems Memory systems in most computing platforms typically consist of two parts: one slow but large memory and one fast but small memory, called **cache**. At each step of computation, a file is requested. If a requested file is available in the cache, then execution starts immediately. Otherwise, a fault is counted to model the delay, etc., and the requested file has to be loaded into the cache before execution. Since cache is of limited size, to load the newly requested file, some existing file has to be ejected from the cache. For fast processing, one needs to minimize the number of faults, which in turn depends on the choice of the file being ejected on each fault by the algorithm. This problem is popularly known as the paging or the caching problem, and is another canonical example of an online problem, since the set of future requested files is unknown and depends on the nature of the computation task. The problem got renewed attention because of widespread use of content distribution networks, e.g., YouTube, that want to store small amounts of popular information close to the user end.

Operations Research/Combinatorial Optimization Suppose the task is to plumb a house, which requires pipes of different lengths. Pipes in the market are, however, available only in fixed lengths. Thus, a relatively basic but non-trivial problem is how to cut the fixed length pipes into required lengths so as to waste as little as possible or to minimize the cost of plumbing. A two-dimensional variant of this example occurs in the production setting where glass/metal sheets are produced in fixed rectangular sizes, rather than in customized forms. Using these fixed size sheets, required sizes are cut, and the objective is to use as few sheets as possible, or to minimize the wastage. This decision problem is also encountered in inventory management,

https://doi.org/10.1017/9781009349178.002 Published online by Cambridge University Press

where pellets or containers need to be packed with objects of different sizes and weights, subject to a size or weight capacity.

The abstracted version of this problem is called **bin-packing**, whose one-dimensional definition is as follows. A large number of bins are available with a finite weight capacity, and items with different weights have to be packed in as few bins as possible, while respecting the bin weight capacity. The obvious generalization of this is when items arrive sequentially, e.g., demand for a certain shape of glass/metal piece or a new shipment, which gives rise to the online setting, where on an item's arrival, the assignment has to be immediate and irrevocable.

Web Advertising One of the most important problems in the online setting, which is also responsible for the renewed interest in the study of online algorithms, is what is called the **AdWords** problem. In the early 2000s, with the explosion of web traffic, web portals had this massive opportunity for revenue generation by displaying advertisements, or ads, to users arriving on their web pages. Each advertiser had a user-specific utility if its ad was viewed by the particular user, for which it was ready to pay a certain price to the web portal, subject to a total payment budget. The web portal's decision problem was to match or assign an advertiser's ad to a slot on the webpage on the user's arrival, so as to maximize the total revenue it could extract from the advertisers. This business model is worth billions of dollars, and solving this problem is highly important.

The AdWords problem is actually a variation of the maximum weight matching problem defined over graphs. For a graph, let each edge have an associated weight. The problem is to choose a subset of edges so as to maximize the sum of weights of the chosen edges, under the *matching* condition that no two chosen edges should have a common endpoint. The more popularly studied version of the matching problem is when the graph is restricted to being a bipartite graph, where there are only two sets of vertices, left and right, and edges exist only between a left and a right vertex, if at all. The bipartite matching problem models many problems of immense interest such as matching customers and taxis for ride hailing applications such as Uber, Ola, etc. The online aspect is immediately clear for the pertinent examples described above, where once a customer is assigned a taxi it cannot be reassigned, and customers arrive with arbitrary destination requests and paying capacity in future.

Networking A network is typically modelled as a graph, and the edges of the graph are either assigned some capacity that limits the amount of traffic through it, or a cost function that is an increasing function of the amount of traffic passing through it. A canonical problem in networking is *routing* with multiple objectives. For example, one may want to maximize the amount of traffic passing through the network respecting edge capacity constraints or minimize the total delay of the network. For this general setup, the most interesting regime is online, since traffic arrives at arbitrary times for arbitrary source–destination pairs, which is difficult to predict. The routing problem is also connected to the fundamental question of online load balancing. To see this, assume that the objective is to minimize the total delay of the network. Clearly, any such solution will make sure that not much traffic passes through any one particular edge, implying balanced traffic across different edges.

Scheduling At a high level, **scheduling** means allocating limited resources to tasks in a certain order. Scheduling problems arise in many different contexts, e.g., machine scheduling on factory floors, computation jobs, queuing systems, airline schedules, and many others. The simplest abstraction of the scheduling problem is that jobs with different processing requirements (typically called sizes) arrive in a queue where they wait to be dispatched to a

https://doi.org/10.1017/9781009349178.002 Published online by Cambridge University Press

server, and depart once their processing is complete. Typical performance metrics studied in scheduling include makespan, total delay or response time, or completion time, which are some specific functions of the difference between the departure and the arrival time of each job. For most of these applications, the natural setting to consider is online, where jobs arrive over time, and decisions about scheduling or job-server assignments have to be made causally without the knowledge of future job arrivals.

Renewable Energy Powered Communication My own interest in studying online algorithms was necessitated by a modern communication problem, where the energy needed for communication is extracted from renewable sources. Traditionally, communication devices have been powered either by conventional energy sources or with batteries. With the increased push for green communication, as well as to increase the usage lifetime for devices installed in the field, harvesting energy from wind, solar, electromagnetic, and kinetic sources, has been envisaged. Even though this basic premise is far reaching, with renewable sources, it is difficult to predict the quantum and arrival times of energy arriving in the future. Initial research on green communication was based on the assumption of certain stochastic models for energy arrivals; however, there was no consensus on what is the 'right' model. To obviate the need for choosing a model, an online communication setting emerges, where an algorithm has to choose its transmission rate or energy usage depending on the energy received so far, without knowing the future energy availability or its distribution. This is a unique online scheduling problem where the resource (energy) needed to process jobs itself arrives over time.

1.3 Metric for Online Algorithms

Recall from (1.1) the definition of the performance metric, called the competitive ratio, for online algorithms. Because of the maximization with respect to the input in (1.1), it is an adversarial metric, where the input can be chosen by an adversary in order to maximize the competitive ratio.

In particular, consider a deterministic online algorithm $\mathcal{A}$. Once the deterministic online algorithm $\mathcal{A}$ is specified, the adversary can choose the input knowing the decisions to be made by $\mathcal{A}$ in order to increase its competitive ratio (1.1) (or decrease (1.2)). For example, for the secretary problem discussed in Section 1.1, if $\mathcal{A}$ is deterministic of the following form: sample the first $n/2$ elements, and record the element with the largest value (called threshold) among the first $n/2$ elements. None of the first $n/2$ elements are selected. Then starting from the $n/2 + 1^{st}$ element, select the earliest element that has a value larger than the threshold. Such an algorithm appears reasonable; however, knowing it, the adversary can ensure that the best element is always part of the first $n/2$ elements. Consequently, the best element will never be selected by $\mathcal{A}$, and $\mu_{\mathcal{A}}$ can be made arbitrarily small (recall that the secretary problem is a maximization problem).

As we will see in many chapters of the book, even under this limitation, there are deterministic online algorithms for many problems with constant competitive ratios.

A natural extension of a deterministic online algorithm is a randomized algorithm, which is essentially equivalent to having a probability distribution over all possible deterministic algorithms. Unlike deterministic online algorithms, for randomized online algorithms, one

https://doi.org/10.1017/9781009349178.002 Published online by Cambridge University Press

needs to be careful about the definition of the adversary: whether the adversary is allowed to see the outcome of the random choices made by the randomized online algorithm or not. This question leads to a broad classification of adversaries, the most popular among them being the oblivious adversary.

Definition 1.3.1 *For randomized online algorithms, the* **oblivious adversary** *has to generate the entire input sequence in advance before any requests are served by the online algorithm. Moreover, the adversary's cost is equal to the cost of the optimal offline algorithm* OPT *for that input sequence.*

To belabour the point, this means that the adversary knows the distribution over all possible deterministic algorithms that the randomized algorithm is using but it cannot see the random choices made by the algorithm sequentially.

For a randomized algorithm $\mathcal{R}$ against an oblivious adversary, the competitive ratio (1.1) specializes to

$$\mu_{\mathcal{R}} = \max_{\sigma} \frac{\mathbb{E}\{f_{\mathcal{R}}(\sigma)\}}{f_{\mathsf{OPT}}(\sigma)}, \tag{1.3}$$

where the expectation is over the random choices made by the algorithm.

There are stronger adversaries that are also defined for randomized online algorithms. We review their definitions and the connections between them in Appendix A.1. Throughout the book, whenever we consider a randomized algorithm, by default it is analysed against an oblivious adversary.

1.3.1 Why Adversarial Inputs

One question that is pertinent while considering online algorithms is: why consider adversarial inputs when *nature* is not adversarial? The choice of adversarial input is primarily to avoid making a choice about the true input model (which may be very difficult). To buttress this point, suppose magically the precise input or a class of inputs is known. In this case, one can design an algorithm that is tuned to that input. However, if the input model needs to be redefined for any reason in future or is inexact, a complete redesign of the algorithm might be necessitated. This reworking can be totally avoided if no assumptions are made on the input. Thus a safe choice is to assume that the input is as bad as it can be for an algorithm compared to OPT, and quantify the performance of the algorithm via its competitive ratio. This approach would have been meaningless if algorithms with reasonable competitive ratios did not exist. However, remarkably, as we see in the book, for a large variety of problems, algorithms with competitive ratios that are constant or that grow logarithmically with the parameters of the input have been designed. This book is dedicated to this success story.

1.3.2 Beyond the Worst Case Input

With the advent of very powerful machine learning algorithms, e.g., neural networks, it is not hard to imagine the setting where future input can in fact be predicted with reasonable accuracy. So the obvious challenge in designing online algorithms is whether we can leverage the prediction information and improve the performance of the algorithm when the prediction is

https://doi.org/10.1017/9781009349178.002 Published online by Cambridge University Press

highly accurate, while still ensuring the worst case guarantee in case the prediction is noisy. This model also alleviates some criticism about the metric of competitive ratio, since its guarantee is against the worst case input; however, by and large, the observed inputs are not adversarial. This new paradigm (called **beyond the worst case**) has attracted sufficient attention from the online algorithms community in the recent past, and in a remarkably short time, algorithms that achieve these dual objectives have been devised for many problems of interest. In this book, we consider beyond the worst case setting for multiple problems, and highlight the main ideas behind this approach.

1.4 Complexity Implications for Online Algorithms

In the usual offline setting for solving any problem, where the full input is available, any useful algorithm should be efficient, i.e., its complexity should be polynomial in the size of the input. It is instructive to note that we have not enforced any such requirement for online algorithms. The reason for this relaxation is that, unlike the offline setting, in the online setting, the adversary can adapt the input sequentially depending on the decisions made by the online algorithm. Thus, it is not easy to find online algorithms with small competitive ratios even with unlimited complexity. For example, for the secretary problem, there is no brute-force or complete enumeration based algorithm that can be shown to have a small competitive ratio. Thus, throughout this book, we will disregard the complexity of the studied online algorithms, where in most of the cases they will be efficient anyway.

1.5 Overview of the Book

In Chapters 2–15, we consider most of the basic and well-known online problems, while in the subsequent chapters, we consider some applied online problems that have applications in many different areas.

For most of the chapters in the book, the focus is on describing the basic ideas that are useful for the considered problem together with elegant analysis. For certain problems, in the interest of the ease of exposition, an algorithm with a weaker competitive ratio guarantee is discussed compared to the best-known algorithm.

We begin the discussion on online algorithms with perhaps the simplest online problem, called the ski-rental problem. With the ski-rental problem, an algorithm has to decide each day between renting (low cost) and buying (high cost) the ski, given the uncertain length of the total skiing season, with the objective of minimizing the total cost paid till the end of the skiing season. The ski-rental problem captures the uncertainty about the future input that an online algorithm faces in the simplest form, and is a remarkably useful primitive for many applied problems. Using the ski-rental problem as an example, we also discuss the generic technique called the Yao's principle to lower bound the competitive ratio of randomized online algorithms.

The list-accessing and bin-packing, which are two of the earliest studied online problems, are discussed next in Chapters 3 and 4. The paging problem, a canonical online problem and

https://doi.org/10.1017/9781009349178.002 Published online by Cambridge University Press

one of the most exhaustively studied in the literature because of its importance in computing systems, is discussed next.

In Chapter 6, we discuss a very general online problem formulation called the metrical task system (MTS) and its special case, the k-server problem. MTS generalizes several other important online problems, such as paging and list-accessing.

Subsequently, the secretary problem and its extensions, the knapsack problem, and the bipartite matching problem are considered. The secretary problem has its origins in the applied probability literature, where exact solutions have been obtained using dynamic programming and the Markov decision theory. The weighted bipartite matching problem is a bit modern and is directly applicable for online web-advertising and auctions.

The classical graph theory problem, the travelling salesman problem, in its online avatar is discussed next, which is useful for unknown graph exploration models. After this, we review the load balancing problem that is of great importance in call centres, supermarket queues, and large-scale cloud servers.

A generic primal–dual technique is discussed in Chapter 10 that is useful for analysing a number of online algorithms for different problems, such as the set cover problem, the bipartite matching problem, and the AdWords problems, to name a few.

In Chapter 11, we discuss the facility location problem, where on arrival of each request, either a new facility is opened (which costs a fixed amount) or the request is assigned to the nearest existing open facility (incurring a distance cost) with an objective of minimizing the total cost after all request arrivals. The k-means clustering problem, which is of immense interest in the machine learning community, is closely related to the facility location problem. We expose this connection and present an online algorithm for the k-means clustering problem and bound its competitive ratio.

Three important variants of the job scheduling problem are presented in Chapters 13–15, where in Chapter 13 we consider that the processor speeds are fixed, while in Chapters 14–15 speed tuneable processors are considered.

In the last part of the book, we consider applied problems, such as multi-commodity routing over an undirected network, server provisioning in large-scale cloud systems, also known as online convex optimization with switching cost, green communication (communication powered with renewable energy sources), and submodular partitioning for welfare maximization.

https://doi.org/10.1017/9781009349178.002 Published online by Cambridge University Press