2

Necessary Background

In this section, we briefly overview the background needed for reading the rest of the survey. In Section 2.1, we discuss linear programming and duality. In Section 2.2, we discuss several classical methods for deriving (offline) approximation algorithms for intractable optimization problems. We demonstrate these ideas on the set-cover problem which we later consider in the online setting. In Section 2.3, we provide basic concepts and definitions related to online computation. This section is not meant to give a comprehensive introduction, but rather only provide the basic notation and definitions used later on in the text. For a more comprehensive discussion of these subjects, we refer the reader to the many excellent textbooks on these subjects. For more information on linear programming and duality, we refer the reader to [42]. For further information on approximation techniques we refer the reader to [90]. Finally, for more details on online computation and competitive analysis we refer the reader to [28].

2.1 Linear Programming and Duality

Linear programming is the problem of minimizing or maximizing a linear objective function over a feasible set defined by a set of linear inequalities. There are several equivalent ways of formulating a linear program. For our purposes, the most convenient one is the following:

$$(P) : \min \sum_{i=1}^{n} c_i x_i \text{ s.t.}$$

For any $1 \leq j \leq m$:

$$\sum_{i=1}^{n} a_{ij} x_i \geq b_j, \quad \forall 1 \leq i \leq n, \quad x_i \geq 0.$$

It is well known that any linear program can be formulated in this way. We refer to such a minimization problem as the primal problem (P). Any vector $x = (x_1, x_2, \ldots, x_n)$ that satisfies all the constraints of (P) is referred to as a feasible solution to the linear program (P). Every primal linear program has a corresponding dual program. The dual linear program of (P) is a maximization linear program: it has m dual variables that correspond to the primal constraints and it has n constraints that correspond to the primal variables. The dual program (D) that corresponds to the linear program formulation (P) is the following:

$$(D) : \max \sum_{j=1}^{m} b_j y_j \text{ s.t.}$$

For any $1 \leq i \leq n$:

$$\sum_{j=1}^{m} a_{ij} y_j \leq c_i, \quad \forall 1 \leq j \leq m, \quad y_j \geq 0.$$

The useful properties of the dual program are summarized in the following theorems.

Theorem 2.1 (Weak duality). Let $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_m)$ be feasible solutions to the primal and the dual linear programs, respectively, then:

$$\sum_{i=1}^{n} c_i x_i \geq \sum_{j=1}^{m} b_j y_j.$$

Weak duality states that the value of any feasible dual solution is at most the value of any feasible primal solution. Thus, the dual program can actually be used as a lower bound for any feasible primal solution. The proof of this theorem is quite simple.

Proof.

$$\sum_{i=1}^{n} c_i x_i \geq \sum_{i=1}^{n} \left(\sum_{j=1}^{m} a_{ij} y_j \right) x_i \tag{2.1}$$

$$= \sum_{j=1}^{m} \left(\sum_{i=1}^{n} a_{ij} x_i \right) y_j \tag{2.2}$$

$$\geq \sum_{j=1}^{m} b_j y_j, \tag{2.3}$$

where inequality (2.1) follows since $y = (y_1, y_2, \dots, y_m)$ is feasible and each x_i is non-negative. Equality (2.2) follows by changing the order of summation. Inequality (2.3) follows since $x = (x_1, x_2, \dots, x_n)$ is feasible and each y_j is non-negative. □

The next theorem is sometimes referred to as the strong duality theorem. It states that if the primal and dual programs are bounded, then the optima of the two programs is equal. The proof of the strong duality theorem is harder and we only state the theorem here without a proof.

Theorem 2.2 (Strong duality). The primal linear program is feasible if and only if the dual linear program has a finite optimal solution. In this case, the value of the optimal solutions of the primal and dual programs is equal.

Finally, we prove an important theorem on approximate complementary slackness which is used extensively in the context of approximation algorithms.

Theorem 2.3 (Approximate complementary slackness). Let $x = (x_1, x_2, \dots, x_n)$ and $y = (y_1, y_2, \dots, y_m)$ be feasible solutions to the primal and dual linear programs, respectively, satisfying the following conditions:

- *Primal complementary slackness*: For $\alpha \geq 1$, for each i, $1 \leq i \leq n$, if $x_i > 0$ then $c_i/\alpha \leq \sum_{j=1}^{m} a_{ij} y_i \leq c_i$.
- *Dual complementary slackness*: For $\beta \geq 1$, for each j, $1 \leq j \leq m$, if $y_j > 0$ then $b_j \leq \sum_{i=1}^{n} a_{ij} x_i \leq b_j \cdot \beta$.

Then

$$\sum_{i=1}^{n} c_i x_i \leq \alpha \cdot \beta \sum_{j=1}^{m} b_j y_j.$$

In particular, if the complementary slackness conditions hold with $\alpha = \beta = 1$, then we get that x and y are both optimal solutions to the primal and dual linear programs, respectively. The proof of the theorem is again very short.

Proof.

$$\begin{aligned}
\sum_{i=1}^{n} c_i x_i &\leq \alpha \sum_{i=1}^{n} \left(\sum_{j=1}^{m} a_{ij} y_i \right) x_i & (2.4)\\
&= \alpha \sum_{j=1}^{m} \left(\sum_{i=1}^{n} a_{ij} x_i \right) y_j & (2.5)\\
&\leq \alpha \cdot \beta \sum_{j=1}^{m} b_j y_j, & (2.6)
\end{aligned}$$

where (2.4) follows from the primal complementary slackness condition. Equality (2.5) follows by changing the order of summation, and Inequality (2.6) follows from the dual complementary slackness condition. □

Theorem 2.3 gives an efficient tool for proving that a solution is approximately optimal. Consider, for example, a minimization problem. Suppose you can find primal and dual solutions that satisfy the (approximate) complementary slackness conditions. Then, the solution for the minimization problem is at most $\alpha \cdot \beta$ times a feasible dual solution. Since, by weak duality (Theorem 2.1), the value of any dual solution is a lower bound on the value of any primal solution, the solution obtained is also at most $\alpha \cdot \beta$ times the optimal primal solution.

Covering/packing linear formulations: A special class of linear programs are those in which the coefficients a_{ij}, b_j and c_i are all non-negative. In this case, the primal program forms a *covering problem* and the dual program forms a *packing problem.* The meaning of these names will become clear in Section 2.2 where we discuss the set cover problem. In the sequel, we sometimes use the notion of *covering–packing* primal–dual pair.

2.2 Approximation Algorithms

In this section, we give a very short background on some basic methods for developing approximation algorithms for intractable problems. Later on we show that similar ideas can be extended and used in the context of online algorithms. We start by formally defining the notions of optimization problems and approximation factors. In an (offline) minimization optimization problem we are given set of instances $\mathbb{I}$. For each instance $I \in \mathbb{I}$ there is a set of feasible solutions. Each feasible solution is associated with a *cost.* Let $\mathrm{OPT}(I)$ be the cost of the minimal feasible solution for instance I. A polynomial time algorithm A is called a c-approximation for a minimization optimization problem if for every instance I it outputs a solution with cost at most $c \cdot \mathrm{OPT}(I)$. The definitions for maximization optimization problems are analogous. In this case, each instance is associated with a *profit.* A c-approximation algorithm is guaranteed to return a solution with cost at least $\mathrm{OPT}(I)/c$, where $\mathrm{OPT}(I)$ is the maximum profit solution.

The set-cover problem: We demonstrate several classic ideas used for developing approximation algorithms via the set cover problem. In the set-cover problem, we are given a set of n elements $X = e_1, e_2, \ldots, e_n$, and a family $\mathbb{S} = s_1, s_2, \ldots, s_m$ of subsets of X, $|\mathbb{S}| = m$. Each set s_j is associated with a non-negative cost c_s. A *cover* is a collection of sets such that their union is X. The objective is to find a cover of

X of minimum cost. This problem is known to be NP-hard. Linear programming relaxations constitute a very useful way for obtaining lower bounds on the optimum value of a given combinatorial problem. To this end, we introduce a non-negative variable x_s for each set $s \in \mathbb{S}$. Initially, we formulate the problem as an integer program, allowing x_s to be either 0 or 1. The following is an integer formulation of the set-cover problem:

$$\min \sum_{s \in \mathbb{S}} c_s x_s \text{ s.t.}$$

For each element e_i $(1 \leq i \leq n)$:

$$\sum_{s | e_i \in S} x_s \geq 1, \quad \forall s \in \mathbb{S}, \quad x_s \in \{0,1\}.$$

It is not hard to verify that the set of feasible solution to this program corresponds to the set of feasible covers. Thus, a minimum cover corresponds to an optimal solution to the integer program. Next, we relax the constraint $x_s \in \{0,1\}$ and get the following linear formulation of the problem:

$$(P) : \min \sum_{s \in \mathbb{S}} c_s x_s \text{ s.t.}$$

For each element e_i $(1 \leq i \leq n)$:

$$\sum_{s | e_i \in S} x_s \geq 1, \quad \forall s \in \mathbb{S}, \quad x_s \geq 0.$$

Since the feasible space of the linear formulation contains all the integral solutions of the integer formulation, we get that the optimal solution to the linear formulation is a lower bound on the value of any integral set cover solution. We also remark that there is no need to demand that $x_s \leq 1$, since a minimum cost solution will never increase the value of any x_s above 1. A solution to the linear formulation is called a fractional set cover solution. In such a relaxed set cover solution, one is allowed to take a fraction x_s of each set and pay only $c_s x_s$ for this fraction. The restriction is that the sum of fractions of sets that contain each element e_i should be at least 1. In the corresponding dual linear program of (P) there is a variable for each element e_i. The dual program (D) is the following:

$$(D) : \max \sum_{e \in X} y_{e_i} \text{ s.t.}$$

For each set $s \in \mathbb{S}$:

$$\sum_{e_i \in S} y_{e_i} \leq c_s, \quad \forall 1 \leq i \leq n, \quad y_{e_i} \geq 0.$$

We next demonstrate several classic approximation techniques using the set cover problem as our running example.

2.2.1 Dual Fitting

We present the dual fitting method by analyzing the greedy algorithm for the set cover problem. The greedy algorithm is depicted below.

Greedy algorithm: Initially, $\mathbb{C} = \emptyset$. Let U be the set of yet uncovered elements.
As long as $U \neq \emptyset$, let $s \in \mathbb{S}$ be the set that minimizes the ratio $c_s/|U \cap s|$:

(1) Add s to $\mathbb{C}$ and set $x_s \leftarrow 1$.
(2) For each $e_i \in (U \cap s)$, $y_{e_i} \leftarrow c_s/|U \cap s|$.

Theorem 2.4. The greedy algorithm is an $O(\log n)$-approximation algorithm for the set-cover problem.

Proof. We are going to show that the algorithm produces throughout its execution both primal and dual solutions. Let P and D be the values of the objective functions of the primal and dual solutions produced, respectively. Initially, $P = D = 0$. We focus on a single iteration of the algorithm and denote by ΔP and ΔD the change in the primal and dual cost, respectively. We prove three simple claims:

(1) The algorithm produces a primal (covering) feasible solution.
(2) In each iteration: $\Delta P \leq \Delta D$.

(3) Each packing constraint in the dual program is violated by a multiplicative factor of at most $O(\log n)$.

The proof follows immediately from the three claims together with weak duality. First, by claim (1) our solution is feasible. By the fact that initially $P = D = 0$ and by claim (2) we get that we produce primal and dual solutions such that $P \leq D$. Finally, by claim (3) we get that dividing the dual solution by $c \log n$, for a constant c, yields a dual feasible solution with value $D' = D/(c \log n)$. Therefore, we get that the primal cost is at most $c \log n$ times a feasible dual solution. Since by weak duality a feasible dual solution is at most the cost of any primal solution, we get that the primal solution is at most $c \log n$ times the optimal (minimal) primal solution.

Proof of (1): Clearly, if there exists a feasible primal solution, then the algorithm will also produce a feasible solution.

Proof of (2): Consider an iteration of the algorithm in which U is the set of uncovered elements and set s is chosen. The change in the primal cost (due to the addition of set s) is c_s. In the dual program, we set the variables corresponding to the $|U \cap s|$ elements covered in the iteration to $c_s/|U \cap s|$. Thus, the total change in the dual profit is also c_s. Note that we only set the dual variable corresponding to each element only once. This happens in the iteration in which the element is covered.

Proof of (3): Consider the dual constraint corresponding to set s. Let $e_1, e_2, \ldots, e_k$ be the elements belonging to set s, sorted with respect to the order they were covered by the greedy algorithm. Consider element e_i; we claim that $y_{e_i} \leq c_s/(k - i + 1)$. This is true since the algorithm chooses the set that minimizes the ratio of cost to number of uncovered elements, which is in fact the value assigned to y_{e_i}. At the time e_i is covered by the greedy algorithm, set s contained at least $k - i + 1$ elements that were still uncovered, thus we get that $y_{e_i} \leq c_s/(k - i + 1)$. Therefore, we get for set s:

$$\sum_{e_i \in s} y_{e_i} \leq \sum_{i=1}^{k} \frac{c_s}{k - i + 1} = H_k \cdot c_s = c_s \cdot O(\log n),$$

where H_k is the kth harmonic number. □

2.2.2 Randomized Rounding

In this section, we design a different $O(\log n)$-approximation algorithm for the set cover problem using a useful technique called *randomized rounding*. The first step is to compute an optimal fractional solution to linear program (P). Then, the fractional solution is rounded by treating the fractions as probabilities. We describe the rounding algorithm slightly different from standard textbooks; however, this description will be useful in the sequel. The rounding algorithm is the following:

```
Randomized rounding algorithm:

  (1) For each set $s \in \mathbb{S}$, choose $2\ln n$ independently random
      variables $X(s,i)$ uniformly at random in the interval $[0,1]$.
  (2) For each set $s$, let $\Theta(s) = \min_{i=1}^{2\ln n} X(s,i)$.
  (3) Solve linear program $(P)$.
  (4) Take set $s$ to the cover if $\Theta(s) \leq x_s$.
```

Theorem 2.5. The algorithm produces a solution with the following properties:

(1) The expected cost of the solution is $O(\log n)$ times the cost of the fractional solution.
(2) The solution is feasible with probability $1 - 1/n > 1/2$.

Since the fractional solution provides a lower bound on any integral solution, we get that the algorithm is an $O(\log n)$-approximation.

Proof. The proof strongly uses the fact that in a feasible solution, for each element, the sum of fractions of the sets containing it is at least 1.

To prove (1) note that for each i, $1 \leq i \leq 2\ln n$, the probability that $X(s,i) \leq x_s$ is exactly x_s. The probability that s is chosen to the solution is the probability that there exists an i, $1 \leq i \leq 2\ln n$, such that $X(s,i) \leq x_s$. Let A_i be the event that $X(s,i) \leq x_s$. Thus, the probability that s is chosen to the solution is the probability of $\bigcup_{i=1}^{2\ln n} A_i$. By the union bound this probability is at most the sum of the probabilities of the different events, which is $2x_s \ln n$. Therefore, using linearity of expectation the expected cost of the solution is at most $2 \ln n$ times the cost of the fractional solution.

To prove (2) pick an element e. Fix any i, $1 \leq i \leq 2 \ln n$. The probability that e is not covered due to i is the probability that none of the variables $X(s,i)$ (for all sets s, $e \in s$) result in choosing a set s covering e. This probability is

$$\prod_{s \in \mathbb{S} | e \in s} (1 - x_s) \leq \exp\left(- \sum_{s \in \mathbb{S} | e \in s} x_s \right) \leq \exp(-1),$$

where the first inequality follows since $1 - x \leq \exp(-x)$. The second inequality follows since each element is fractionally covered. Since we choose $2 \ln n$ random variables independently, the probability that e is not covered is at most $\exp(-2 \ln n) = 1/n^2$. Using the union bounds we get that the probability that there exists an element e which is not covered is at most $n \cdot 1/n^2 = 1/n$. □

2.2.3 The Primal–Dual Schema

In this section, we give a third approximation algorithm for the set cover problem which is based on the primal–dual schema. The algorithm is the following:

```
Primal–dual algorithm:
While there exists an uncovered element $e_i$:

  (1) Increase the dual variable $y_{e_i}$ continuously.
  (2) If there exists a set $s$ such that $\sum_{e \in s} y_e = c_s$: take $s$ to the
      cover and set $x_s \leftarrow 1$.
```

Let f be the maximum frequency of an element (i.e., the maximum number of sets an element belongs to).

Theorem 2.6. The algorithm is an f-approximation for the set-cover problem.

Proof. Clearly, the primal solution produced by the algorithm is feasible, since we pick sets to the cover as long as the solution is infeasible. The dual solution produced by the algorithm is also feasible, since set s is taken to the solution only when the dual constraint corresponding to it becomes tight. From then on, we never increase a dual variable corresponding to an element belonging to s. Finally, the primal complementary slackness condition holds with $\alpha = 1$, since, if $x_s > 0$, then $\sum_{e \in s} y_{e_i} = c_s$. The dual complementary slackness condition holds with $\beta = f$, since, if $y_{e_i} > 0$, then $1 \leq \sum_{s|e \in s} x_s \leq f$. Thus, by Theorem 2.3, we get that the algorithm is an f-approximation. □

Remark 2.7. For the special case of the vertex cover problem this algorithm is a 2-approximation.

2.3 Online Computation

The traditional design and analysis of algorithms assumes that complete knowledge of the entire input is available to an algorithm. However, this is not the case in an online problem, where the input is revealed in parts, and an online algorithm is required to respond to each new input upon arrival. Previous decisions of the online algorithm cannot be revoked. Thus, the main issue in online computation is obtaining good performance in the face of uncertainty, since the "future" is unknown to the algorithm. A standard measure by now for evaluating the performance of an online algorithm is the *competitive ratio*, which compares the performance of an online algorithm to that of an offline algorithm which is given the whole input sequence beforehand.

The precise definition of the competitive factor is the following. Suppose we are given a minimization optimization problem I. For each instance of I there is a set of feasible solutions. Each feasible solution is associated with a *cost*. Let $\mathrm{OPT}(I)$ be the optimal cost of a feasible solution for instance I. In the online case, the instance is given to the algorithm in parts. An online algorithm A is said to be c-competitive for I if for every instance of I it outputs a solution of cost at most $c \cdot \mathrm{OPT}(I) + \alpha$, where the additive term α is independent of the request sequence. if $\alpha = 0$ then the algorithm is called *strictly* c-competitive. We are not going to distinguish between the two notions. The definition of competitiveness of maximization optimization problems is analogous. When considering a maximization problem each instance is associated with a *profit.* A c-competitive algorithm is guaranteed to return a solution with cost at least $\mathrm{OPT}(I)/c - \alpha$, where $\mathrm{OPT}(I)$ is the maximum profit solution, and α is an additive term independent of the request sequence.

A common concept in competitive analysis is that of an *adversary.* The online solution can be viewed as a game between an online algorithm and a malicious adversary. While the online algorithm's strategy is to minimize its cost, the adversary's strategy is to construct the worst possible input for the algorithm. Using this view, the adversary produces a sequence $\sigma = \sigma_1, \sigma_2, \ldots$ of requests that define the instance I. A c-competitive online algorithm should then be able to produce a solution of cost no more than c times $\mathrm{OPT}(\sigma)$ for every request sequence σ.

There are several known natural models of adversaries in the context of randomized online algorithms. In this work, we only consider a model in which the adversary knows the online algorithm, as well as the probability distribution used by the online algorithm to make its random decisions. However, the adversary is unaware of the actual random choices made by the algorithm throughout its execution. This kind of adversary is called an *oblivious adversary.* A randomized online algorithm is c-competitive against an oblivious adversary, if for every request sequence σ, the expected cost of the algorithm on σ is at most $c \cdot \mathrm{OPT}(\sigma) + \alpha$. The expectation is taken over all random choices made by the algorithm. Since the oblivious adversary has no information about the actual random choices made by the algorithm, the sequence σ can be constructed ahead of time and $\mathrm{OPT}(\sigma)$ is not a random variable. In the sequel, whenever we have a randomized online algorithm, we simply say that it is c-competitive, and mean that it is c-competitive against an oblivious adversary. Again, the definitions for maximization problems are analogous.

2.4 Notes

The dual-fitting analysis in Section 2.2.1 of the greedy heuristic is due to [78, 41]. The algorithm in Section 2.2.3 is due to Bar-Yehuda and Even [17]. The set-cover problem is a classic NP-hard problem that was studied extensively in the literature. The best approximation factor achievable for it in polynomial time (assuming $P \neq \text{NP}$) is $\Theta(\log n)$ [41, 47, 68, 78].

The introduction here is only meant to establish basic notation and terminology for the rest of our discussion. The area of linear programming, duality, approximation algorithms, and online computation have been studied extensively. For more information on linear programming and duality we refer the reader to [42]. For further information on approximation techniques we refer the reader to [90]. Finally, for more details on online computation and competitive analysis we refer the reader to [28].

3 A First Glimpse: The Ski Rental Problem

Let us start with a classic online problem, *ski rental*, through which we will demonstrate the basic ideas underlying the online primal–dual approach. At a ski resort renting skis costs \$1 per day, while buying skis costs \$$B$. A skier arrives at the ski resort for a ski vacation and has to decide whether to rent or buy skis. However, an unknown factor is the number of remaining skiing days that are left before the snow melts. (We should note that this is the skier's last vacation.) In spite of its apparent simplicity, the ski rental problem captures the essence of online rent or buy dilemmas. The ski rental problem is well understood. There exists a simple deterministic 2-competitive algorithm for the problem and a randomized $e/(e-1)$-competitive algorithm. Both results are tight. We show here how to obtain these results using a primal–dual approach.

The first step towards obtaining an online primal–dual algorithm is formulating the problem in hand as a linear program. Since the offline ski rental problem is so simple, casting it as a linear program may seem a bit unnatural. However, this formulation turns out to be very useful. We define an indicator variable x which is set to 1 if the skier buys the skis. For each day j, $1 \leq j \leq k$, we define an indicator variable z_j which is set to 1 if the skier decides to rent skis on day j. The constraints guarantee that on each day we either rent skis or buy them. This gives us the following integer formulation for the ski rental problem:

$$\min B \cdot x + \sum_{j=1}^{k} z_j$$

For each day j:

$$x + z_j \geq 1, \quad x \in \{0,1\}, \ \forall j, z_j \in \{0,1\}.$$

We next relax the problem and allow x and each z_j to be in $[0,1]$. The linear program is given in Figure 3.1 (the primal program). Note that there is always an optimal integral solution, and thus the relaxation has no integrality gap. The dual program is also extremely simple and has a variable y_j corresponding to each day j. We then have a constraint $y_j \leq 1$ that corresponds to each primal variable z_j, and a constraint $\sum_{j=1}^{k} y_j \leq B$ that corresponds to the primal variable x. Note that the linear programming formulation forms a covering–packing primal–dual pair.

Next, consider the online scenario in which k (the number of ski days) is unknown in advance. This scenario can be captured in the linear formulation in a very natural way. Whenever we have a new ski day, the primal linear program is updated by adding a new covering constraint. The dual program is updated by adding a new dual variable which is added to the packing constraints. The online requirement is that previous decisions cannot be undone. That is, if we already rented skis yesterday, we cannot change this decision today. This requirement is captured in the primal linear program by the restriction that the primal variables are monotonically non-decreasing over time.

Dual (Packing)		Primal (Covering)	
Maximize:	$\sum_{j=1}^{k} y_j$	Minimize :	$B \cdot x + \sum_{j=1}^{k} z_j$
Subject to:		Subject to:	
	$\sum_{j=1}^{k} y_j \leq B$	For each day j:	$x + z_j \geq 1$
For each day j:	$0 \leq y_j \leq 1$		$x \geq 0, \forall j, z_j \geq 0$

Fig. 3.1 The fractional ski problem (the primal) and the corresponding dual problem.

Obtaining an optimal deterministic online algorithm for the ski rental problem is very simple. First, rent the skis for the first $B - 1$ days, and then buy them on the Bth day. If $k < B$, the algorithm is optimal. If $k \geq B$, then the algorithm spends \$$2B$ dollars, while the optimal solution buys skis on the first day and spends only \$$B$ dollars. Thus, the algorithm is 2-competitive. This simple algorithm and analysis can be obtained in a somewhat less natural way using a primal–dual analysis. On the jth day, a new primal constraint $x + z_j \geq 1$ and a new dual variable y_j arrive. If the primal constraint is already satisfied, then do nothing. Otherwise, increase y_j continuously until some dual constraint becomes tight. Set the corresponding primal variable to be 1. The above algorithm is a simple application of the primal–dual schema, and its analysis is very simple using the approximate complementary slackness conditions: If $y_j > 0$ then $1 \leq x + z_j \leq 2$. Moreover, if $x > 0$ then $\sum_{j=1}^{k} y_j = B$, and if $z_j > 0$ then $y_j = 1$. Thus, by Theorem 2.3, the algorithm is 2-competitive. It is not hard to see that both of the above algorithms are actually equivalent.

Developing an optimal randomized algorithm is not as straightforward as the deterministic one, yet the primal–dual approach turns out to be very useful towards this end. The first step is designing a deterministic fractional competitive algorithm. Recall that in the fractional case the primal variables can be in the interval $[0, 1]$, and the variables are required to be monotonically non-decreasing over time, during the execution of the algorithm. The online algorithm is the following:

```
(1) Initially, $x \leftarrow 0$.
(2) Each new day ($j$th new constraint), if $x < 1$:
    (a) $z_j \leftarrow 1 - x$.
    (b) $x \leftarrow x(1 + 1/B) + 1/(c \cdot B)$.   (The value of $c$ will
        be determined later.)
    (c) $y_j \leftarrow 1$.
```

The analysis is simple. We show that:

(1) The primal and dual solutions are feasible.
(2) In each iteration (day), the ratio between the change in the primal and dual objective functions is bounded by $(1 + 1/c)$.

Using the weak duality theorem (Theorem 2.1) we immediately conclude that the algorithm is $(1 + 1/c)$-competitive.

The proof is very simple. Denote the number of ski days by k. First, since we set $z_j = 1 - x$ (whenever $x < 1$), the primal solution produced is feasible. To show feasibility of the dual solution, we need to show that

$$\sum_{j=1}^{k} y_j \leq B.$$

We prove this by showing that $x \geq 1$ after at most B days of ski. Denote the increments of x (in each day) by $x_1, x_2, \ldots, x_k$, where $x = \sum_{j=1}^{k} x_j$. It can be easily seen that $x_1, x_2, \ldots, x_k$ is a geometric sequence, defined by $x_1 = 1/(cB)$ and $q = 1 + 1/B$. Thus, after B days,

$$x = \frac{\left(1 + \frac{1}{B}\right)^B - 1}{c}.$$

Setting $c = (1 + 1/B)^B - 1$ guarantees that $x = 1$ after B days.

Second, if $x < 1$, in each iteration the dual objective function increases by 1, and the increase in the primal objective function is $B\Delta x + z_j = x + 1/c + 1 - x = 1 + 1/c$, and thus the ratio is $(1 + 1/c)$. Concluding, the competitive ratio is $1 + 1/c \approx e/(e - 1)$ for $B \gg 1$.

Converting the fractional solution into a randomized competitive algorithm with the same competitive ratio is easy. We arrange the increments of x on the $[0, 1]$ interval and choose ahead of time (before executing the algorithm) $\alpha \in [0, 1]$ uniformly in random. We are going to buy skis on the day corresponding to the increment of x to which α belongs.

We now analyze the expected cost of the randomized algorithm. Since the probability of buying skis on the jth day is equal to x_j, the expected cost of buying skis is precisely $B \cdot \sum_{j=1}^{k} x_j = Bx$, which is exactly the first term in the primal objective function. The probability of renting skis on the jth day is equal to the probability of *not* buying skis on or before the jth day, which is $1 - \sum_{i=1}^{j} x_i$. Since

$$z_j = 1 - \sum_{i=1}^{j-1} x_i \geq 1 - \sum_{i=1}^{j} x_i,$$

we get that the probability of renting on the jth day is at most z_j, corresponding to the second term in the primal objective function. Thus, by linearity of expectation, for any number of ski days, the expected cost of the randomized algorithm is at most the cost of the fractional solution.

3.1 Notes

The results in this chapter are based on the work of Buchbinder et al. [30] who showed how to obtain a randomized $e/(e-1)$-competitive algorithm via the primal–dual approach. The randomized $e/(e-1)$-competitive factor for the ski rental problem was originally obtained by Karlin et al. [71]. The role of randomization in the ski problem was later restudied, along with other problems, in [70]. The deterministic 2-competitive algorithm is due to [72].

4

The Basic Approach

We are now ready to extend the ideas used in the previous chapter for the ski rental problem and develop a general recipe for online problems which can be formulated as packing–covering linear programs. In Section 4.1, we formally define a general online framework for packing–covering problems. In Section 4.2, we develop several competitive online algorithms for this framework. In Section 4.3, we prove lower bounds and show that these algorithms are optimal. Finally, in Section 4.4, we give two simple examples that utilize the new ideas developed here.

4.1 The Online Packing–Covering Framework

We are going to define here a general online framework for packing–covering problems. Let us first consider an "offline" covering linear problem. The objective is to minimize the total cost given by a linear cost function $\sum_{i=1}^{n} c_i x_i$. The feasible solution space is defined by a set of m linear constraints of the form $\sum_{i=1}^{n} a(i,j) x_i \geq b(j)$, where the entries $a(i,j), b(j)$ and c_i are *non-negative*. For simplicity, we consider in this chapter a simpler setting in which $b(j) = 1$ and $a(i,j) \in \{0,1\}$. In Section 14, we show how to extend the ideas we present here to handle general (non-negative) values of $a(i,j)$ and $b(j)$. In the simpler setting, each covering constraint j can be associated with a set $S(j)$ such that $i \in S(j)$ if $a(i,j) = 1$. The jth covering constraint then reduces to simply $\sum_{i \in S(j)} x_i \geq 1$. Any primal covering instance has a corresponding dual packing problem that provides a lower bound on any feasible solution to the instance. A general form of a (simpler) primal covering problem along with its dual packing problem is given in Figure 4.1.

The *online covering problem* is an online version of the covering problem. In the online setting, the cost function is known in advance, but the linear constraints that define the feasible solution space are given to the algorithm one-by-one. In order to maintain a feasible solution to the current set of given constraints, the algorithm is allowed to increase the variables. It may not, however, decrease any previously increased variable. The objective of the algorithm is to minimize the objective function. The reader may already have noticed that this online setting captures the ski rental problem (Chapter 3) as a special case.

The *online packing problem* is an online version of the packing problem. In the online setting, the values c_i $(1 \leq i \leq n)$ are known in advance. However, the profit function and the exact packing constraints are not known in advance. In the jth round, a new variable y_j is introduced, along with the set of packing constraints it appears in. The algorithm may increase the value of a variable y_j only in the round in which it is given, and may not decrease or increase the values of any previously given variables. Note that variables that have not yet been introduced may also later appear in the same packing constraints. This actually means that each packing constraint is revealed to the algorithm gradually. The objective of the algorithm is to maximize the objective function while maintaining the feasibility of all packing constraints. Although this online setting seems at first glance a bit

(P): Primal (Covering)		(D): Dual (Packing)	
Minimize:	$\sum_{i=1}^{n} c_i x_i$	Maximize:	$\sum_{j=1}^{m} y_j$
subject to:		subject to:	
$\forall 1 \leq j \leq m$:	$\sum_{i \in S(j)} x_i \geq 1$	$\forall 1 \leq i \leq n$:	$\sum_{j \mid i \in S(j)} y_j \leq c_i$
$\forall 1 \leq i \leq n$:	$x_i \geq 0$	$\forall 1 \leq j \leq m$:	$y_j \geq 0$

Fig. 4.1 Primal (covering) and dual (packing) problems.

unnatural, we later show that many natural online problems reduce to this online setting.

Clearly, at any point of time the linear constraints that have appeared so far define a *sub-instance* of the final covering instance. The dual packing problem of this sub-instance is a *sub-instance* of the final dual packing problem. In the dual packing sub-instance, only part of the dual variables are known, along with their corresponding coefficients. Thus, the two sub-instances derived from the above online settings form a *primal–dual pair*.

The algorithms we propose in the next section maintain at each step solutions for both the primal and dual sub-instances. When a new constraint appears in the online covering problem, our algorithms also consider the new *corresponding* dual variable and its coefficients. Conversely, when a new variable along with its coefficients appear in the online fractional packing problem, our algorithms also consider the *corresponding* new constraint in the primal sub-instance.

4.2 Three Simple Algorithms

In this section, we present three algorithms with the same (optimal) performance guarantees for the online covering/packing problem. Although the performance of all three algorithms in the worst case is the same, their properties do vary, and thus certain algorithms are better suited for particular applications. Also, the ideas underlying each of the algorithms can be extended later to more complex settings. The first algorithm is referred to as the *basic discrete algorithm* and is a direct extension of the algorithm for the ski rental in Section 3. The algorithm is the following:

```
Algorithm 1:
Upon arrival of a new primal constraint $\sum_{i \in S(j)} x_i \geq 1$ and the corresponding dual variable $y_j$:
    (1) While $\sum_{i \in S(j)} x_i < 1$:
        (a) For each $i \in S(j)$: $x_i \leftarrow x_i(1 + 1/c_i) + 1/(|S(j)|c_i)$.
        (b) $y_j \leftarrow y_j + 1$.
```

We assume that each $c_i \geq 1$. This assumption is not restrictive and we discuss the reason for that later on. Let $d = \max_j |S(j)| \leq m$ be the maximum "size" of a covering constraint. We prove the following theorem:

Theorem 4.1. The algorithm produces:

- A (fractional) covering solution which is $O(\log d)$-competitive.
- An "integral" packing solution which is 2-competitive and violates each packing constraint by at most a factor of $O(\log d)$.

We remark that it is possible to obtain a feasible packing solution by scaling the update of each y_j by a factor of $O(\log\ d)$. This, however, will yield a non-integral packing solution. It is also beneficial for the reader to note the similarity ("in spirit") between the proof of Theorem 4.1 and the dual fitting based proof of the greedy heuristic for the set cover problem in Section 2.2.1.

Proof. Let P and D be the values of the objective function of the primal and the dual solutions the algorithm produces, respectively. Initially, $P = D = 0$. The dual variables start from zero and increase in increments of one unit, the therefore the dual solution is integral.

Let ΔP and ΔD be the changes in the primal and dual cost, respectively, in a particular iteration of the algorithm in which we execute the inner loop. We prove three simple claims:

(1) The algorithm produces a primal (covering) feasible solution.
(2) In each iteration: $\Delta P \leq 2\Delta D$.
(3) Each packing constraint in the dual program is violated by at most $O(\log d)$.

The proof then follows immediately from the three claims together with weak duality.

Proof of (1): Consider a primal constraint $\sum_{i \in S(j)} x_i \geq 1$. During the jth iteration the algorithm increases the values of the variables x_i until the constraint is satisfied. Subsequent increases of the variables cannot result in infeasiblity.

Proof of (2): Whenever the algorithm updates the primal and dual solutions, the change in the dual profit is 1. The change in the primal cost is

$$\sum_{i \in S(j)} c_i \Delta x_i = \sum_{i \in S(j)} c_i \left(\frac{x_i}{c_i} + \frac{1}{|S(j)| c_i} \right) = \sum_{i \in S(j)} \left(x_i + \frac{1}{|S(j)|} \right) \leq 2,$$

where the final inequality follows since the covering constraint is infeasible at the time of the update.

Proof of (3): Consider any dual constraint $\sum_{j | i \in S(j)} y_j \leq c_i$. Whenever we increase a variable y_j, $i \in S(j)$, by one unit we also increase the variable x_i in line (1a). We prove by simple induction that the variable x_i is bounded from below by the sum of a geometric sequence with $a_1 = 1/(dc_i)$ and $q = (1 + 1/c_i)$. That is,

$$x_i \geq \frac{1}{d} \left(\left(1 + \frac{1}{c_i} \right)^{\sum_{j | i \in S(j)} y_j} - 1 \right). \tag{4.1}$$

Initially, $x_i = 0$, so the induction hypothesis holds. Next, consider an iteration in which variable y_k increases by 1. Let $x_i(\text{start})$ and $x_i(\text{end})$ be the values of variable x_i before and after the increment, respectively. Then,

$$\begin{aligned} x_i(\text{end}) &= x_i(\text{start}) \left(1 + \frac{1}{c_i} \right) + \frac{1}{|S(j)| c_i} \\ &\geq x_i(\text{start}) \left(1 + \frac{1}{c_i} \right) + \frac{1}{d \cdot c_i} \\ &\geq \frac{1}{d} \left(\left(1 + \frac{1}{c_i} \right)^{\sum_{j | i \in S(j) \setminus \{k\}} y_j} - 1 \right) \left(1 + \frac{1}{c_i} \right)^{y_k} + \frac{1}{d \cdot c_i} \\ &= \frac{1}{d} \left(\left(1 + \frac{1}{c_i} \right)^{\sum_{j | i \in S(j)} y_j} - 1 \right). \end{aligned}$$

Note that the inductive hypothesis is applied to $x_i(\text{start})$.

Next, observe that the algorithm never updates any variable $x_i \geq 1$ (since it cannot belong to any unsatisfied constraint). Since each $c_i \geq 1$

and $d \geq 1$, we have that $x_i < 1(1+1)+1 = 3$. Together with inequality (4.1) we get that:

$$3 \geq x_i \geq \frac{1}{d}\left(\left(1+\frac{1}{c_i}\right)^{\sum_{j|i\in S(j)} y_j} - 1\right).$$

Using again the fact that $c_i \geq 1$ and simplifying we get the desired result:

$$\sum_{j|i\in S(j)} y_j \leq c_i \log_2(3d+1) = c_i \cdot O(\log d).$$

□

The basic discrete algorithm is extremely simple and we show in the sequel its many applications. We are now going to derive a slightly different algorithm, which has a continuous flavor and is more in the spirit of the primal–dual schema. This algorithm will also be of guidance for gaining intuition about the right relationship between primal and dual variables. The algorithm is the following:

```
Algorithm 2:
Upon arrival of a new primal constraint $\sum_{i\in S(j)} x_i \geq 1$ and the corresponding dual variable $y_j$:

    (1) While $\sum_{i\in S(j)} x_i < 1$:
        (a) Increase the variable $y_j$ continuously.
        (b) For each variable $x_i$ that appears in the (yet unsatisfied) primal constraint increase $x_i$ according to the following function:
            $x_i \leftarrow \frac{1}{d}\left[\exp\left(\frac{\ln(1+d)}{c_i}\sum_{j|i\in S(j)} y_j\right) - 1\right].$
```

Note that the exponential function for variable x_i contains dual variables that correspond to future constraints. However, these variables are all initialized to 0, so they do not contribute to the value of the function. Although the algorithm is described in a continuous fashion, it is not hard to implement it in a discrete fashion in any desired accuracy. We discuss the intuition of the exponential function we use after proving the following theorem:

Theorem 4.2. The algorithm produces:

- A (fractional) covering solution which is feasible and $O(\log d)$-competitive.
- A (fractional) packing solution which is feasible and $O(\log d)$-competitive.

Proof. Let P and D be the values of the objective function of the primal and the dual solutions produced by the algorithm, respectively. Initially, $P = D = 0$. We prove three simple claims:

(1) The algorithm produces a primal (covering) feasible solution.
(2) In each iteration j: $\partial P/\partial y_j \leq 2\ln(1+d) \cdot \partial D/\partial y_j$.
(3) Each packing constraint in the dual program is feasible.

The theorem then follows immediately from the three claims together with weak duality.

Proof of (1): Consider a primal constraint $\sum_{i \in S(j)} x_i \geq 1$. During the iteration in which the the jth primal constraint and dual variable y_j appear, the algorithm increases the values of the variables x_i until the constraint is satisfied. Subsequent increases of variables cannot result in infeasibility.

Proof of (2): Whenever the algorithm updates the primal and dual solutions, $\partial D/\partial y_j = 1$. The derivative of the primal cost is

$$\begin{aligned} \frac{\partial P}{\partial y_j} &= \sum_{i \in S(j)} c_i \frac{\partial x_i}{\partial y_j} \\ &= \sum_{i \in S(j)} c_i \left(\frac{\ln(1+d)}{c_i} \frac{1}{d} \left[\exp\left(\frac{\ln(1+d)}{c_i} \sum_{j | i \in S(j)} y_j \right) \right] \right) \end{aligned}$$

$$= \ln(1+d) \sum_{i \in S(j)} \left(\frac{1}{d} \left[\exp\left(\frac{\ln(1+d)}{c_i} \sum_{j|i \in S(j)} y_j \right) - 1 \right] + \frac{1}{d} \right)$$

$$= \ln(1+d) \sum_{i \in S(j)} \left(x_i + \frac{1}{d} \right) \leq 2\ln(1+d). \qquad (4.2)$$

The last inequality follows since the covering constraint is infeasible.

Proof of (3): Consider any dual constraint $\sum_{j|i \in S(j)} y_j \leq c_i$. The corresponding variable x_i is always at most 1, since otherwise it cannot belong to any unsatisfied constraint. Thus, we get that:

$$x_i = \frac{1}{d} \left[\exp\left(\frac{\ln(1+d)}{c_i} \sum_{j|i \in S(j)} y_j \right) - 1 \right] \leq 1.$$

Simplifying we get that:

$$\sum_{j|i \in S(j)} y_j \leq c_i.$$

□

Discussion: As can be seen from the proof, the basic discrete algorithm and the continuous algorithm are essentially the same, since $(1 + 1/c_i)$ is approximately $\exp(1/c_i)$. The function in the continuous algorithm is then approximated by Inequality (4.1) in Theorem 4.1. The approximate equality is true as long as c_i is not too small, which is why the assumption that $c_i \geq 1$ is needed in the discrete algorithm. In addition, the discrete algorithm allows the primal variables to exceed the value of 1, which is unnecessary (and can easily be avoided). For these reasons, the proof of the continuous algorithm is a bit simpler. However, in contrast, the description of the discrete algorithm is simpler and more intuitive. Also, in the discrete algorithm, it is not necessary to know the value of d in advance (as long as it is not needed to scale down the dual variables to make the dual solution feasible).

The reader may wonder at this point how did we choose the function used in the algorithm for updating the primal and dual variables. We will try to give here a systematic way of deriving this function. Consider the point in time in which the jth primal constraint is given and assume that it is not satisfied. Our goal is to bound the derivative of the primal cost (denoted by P) as a function of the dual profit (denoted by D). That is, show that

$$\frac{\partial P}{\partial y_j} = \sum_{i \in S(j)} c_i \frac{\partial x_i}{\partial y_j} \leq \alpha \frac{\partial D}{\partial y_j},$$

where α is going to be the competitive factor. Suppose that the derivative of the primal cost satisfies:

$$\sum_{i \in S(j)} c_i \frac{\partial x_i}{\partial y_j} = A \sum_{i \in S(j)} \left(x_i + \frac{1}{d} \right). \tag{4.3}$$

Then, since $\sum_{i \in S(j)} x_i \leq 1$, $\sum_{i \in S(j)} 1/d \leq 1$, and $\partial D / \partial y_j = 1$, we get that

$$A \sum_{i \in S(j)} \left(x_i + \frac{1}{d} \right) \leq 2A \frac{\partial D}{\partial y_j}.$$

Thus, $\alpha = 2A$. Now, satisfying equality (4.3) requires solving the following differential equation for each $i \in S(j)$:

$$\frac{\partial x_i}{\partial y_j} = \frac{A}{c_i} \left(x_i + \frac{1}{d} \right).$$

It is easy to verify that the solution is a family of functions of the following form:

$$x_i = B \cdot \exp\left(\frac{A}{c_i} \sum_{\ell \ | i \in S(j)} y_\ell \right) - \frac{1}{d},$$

where B can take on any value. Next, we have the following two boundary conditions on the solution:

- Initially, $x_i = 0$, and this happens when $1/c_i \sum_{j | i \in S(j)} y_j = 0$.
- If $1/c_i \sum_{j | i \in S(j)} y_j = 1$, (i.e., the dual constraint is tight), then $x_i = 1$. (Then, the primal constraint is also satisfied.)

The first boundary condition gives $B = 1/d$. The second boundary condition gives us $A = \ln(d+1)$. Putting everything together we get the exact function used in the algorithm.

We next describe a third algorithm. This algorithm is also continuous, yet different from the previous one. The idea is to utilize the complementary slackness conditions in the online algorithm. This idea turns out to be useful in some applications we discuss in later chapters. Again, let $d = \max_j |S(j)| \leq m$ be the maximum size of a constraint. The description of the algorithm is the following:

Algorithm 3:
Upon arrival of a new primal constraint $\sum_{i \in S(j)} x_i \geq 1$ and the corresponding dual variable y_j:

```
(1) While $\sum_{i \in S(j)} x_i < 1$:
    (a) Increase variable $y_j$ continuously.
    (b) If $x_i = 0$ and $\sum_{j|i \in S(j)} y_j = c_i$ then set $x_i \leftarrow 1/d$.
    (c) For each variable $x_i$, $1/d \leq x_i < 1$, that appears in
        the (yet unsatisfied) primal constraint, increase $x_i$
        according to the following function:
```

$$x_i \leftarrow \frac{1}{d} \exp\left(\frac{\sum_{j|i \in S(j)} y_j}{c_i} - 1 \right).$$

First, note that the exponential function equals $1/d$ when $\sum_{j|i \in S(j)} y_j = c_i$ and so the algorithm is well defined. We next prove the following theorem:

Theorem 4.3. The algorithm produces:

- A (fractional) $O(\log d)$-competitive covering solution.
- A (fractional) 2-competitive packing solution and violates each packing constraint by a factor of at most $O(\log d)$.

We remark that similarly to the basic discrete algorithm, it is possible to make the packing solution feasible (and $O(\log d)$-competitive) by scaling down each y_j by a factor of $O(\log d)$.

Proof. Let P and D be the values of the objective function of the primal and dual solutions, respectively. Initially, $P = D = 0$. We prove three simple claims:

(1) The algorithm produces a primal (covering) feasible solution.
(2) Each packing constraint in the dual program is violated by a factor of at most $O(\log d)$.
(3) $P \leq 2D$.

The theorem then follows immediately from the three claims together with weak duality.

Proof of (1): Consider a primal constraint $\sum_{i \in S(j)} x_i \geq 1$. During the jth iteration the algorithm increases the values of the variables x_i until the constraint is satisfied. Subsequent increases of the variables cannot result in infeasibility.

Proof of (2): Consider any dual constraint $\sum_{j|i \in S(j)} y_j \leq c_i$. The corresponding variable x_i cannot exceed 1, since otherwise it would not belong to any unsatisfied constraint. Thus, we get that:

$$x_i = \frac{1}{d} \exp\left(\frac{\sum_{j|i \in S(j)} y_j}{c_i} - 1 \right) \leq 1.$$

Simplifying, we get that

$$\sum_{j|i \in S(j)} y_j \leq c_i(1 + \ln d).$$

Proof of (3): We partition the contribution to the primal objective function into two parts. Let C_1 be the contribution to the primal cost from Step (1b), due to the increase of primal variables from 0 to $1/d$. Let C_2 be the contribution to the primal cost from Step (1c) of the algorithm. It is also beneficial for the reader to observe the similarity between the arguments used for bounding C_1 and those used for the proof of the approximation factor of the primal–dual algorithm in Section 2.2.3.

Bounding C_1: Let $\tilde{x}_i = \min(x_i, 1/d)$. Our goal is to bound $\sum_{i=1}^{n} c_i \tilde{x}_i$. To do this we observe the following. First, the algorithm guarantees that if $x_i > 0$, and therefore $\tilde{x}_i > 0$, then:

$$\sum_{j|i\in S(j)} y_j \geq c_i \quad \text{(primal complementary slackness)} \tag{4.4}$$

Next, if $y_j > 0$, then:

$$\sum_{i\in S(j)} \tilde{x}_i \leq 1 \quad \text{(dual complementary slackness)} \tag{4.5}$$

Inequality (4.5) follows since $\tilde{x}_i \leq 1/d$. Thus, even if for all i, $\tilde{x}_i = 1/d \leq 1/|S(j)|$, then $\sum_{i\in S(j)} \tilde{x}_i \leq 1$. Note that the inequality holds for any y_j, regardless if $y_j > 0$. By the primal and dual complementary slackness conditions we get that:

$$\begin{aligned}
\sum_{i=1}^{n} c_i\tilde{x}_i &\leq \sum_{i=1}^{n} \left(\sum_{j|i\in S(j)} y_j \right) \tilde{x}_i && (4.6)\\
&= \sum_{j=1}^{m} \left(\sum_{i\in S(j)} \tilde{x}_i \right) y_j && (4.7)\\
&\leq \sum_{j=1}^{m} y_j, && (4.8)
\end{aligned}$$

where Inequality (4.6) follows from Inequality (4.4). Equality (4.7) follows by changing the order of summation. Inequality (4.8) follows from Inequality (4.5). Thus, we get that C_1 is at most the dual cost.

Bounding C_2: Whenever the algorithm updates the primal and dual solutions, $\partial D/\partial y_j = 1$. It is easy to verify that the derivative of the primal cost is:

$$\frac{\partial P}{\partial y_j} = \sum_{i\in S(j)} c_i \frac{\partial x_i}{y_j} = \sum_{i\in S(j)} c_i \frac{x_i}{c_i} \leq 1. \tag{4.9}$$

The last inequality follows since the covering constraint is infeasible at the update time. Thus, C_2 is also bounded from above by the dual cost. We conclude: $P = C_1 + C_2 \leq 2D$. □

Remark 4.4. It is possible to even further optimize the performance of Algorithm 3. Initialize each x_i to $\mu = 1/d \ln d$ (instead of $1/d$), and change the continuous update rule to be $x_i \leftarrow \mu \exp\left(\sum_{j|i \in S(j)} y_j/c_i - 1\right)$. This will guarantee that $C_1 + C_2 = (1 + 1/\ln d) \cdot D$, while the dual solution is violated by a factor of $1 + \ln d + \ln\ln d$, yielding that the competitive ratio is $\ln d + \ln\ln d + O(1)$. It is also possible to prove a corresponding lower bound of H_d on the competitive ratio of any online algorithm, meaning that the competitive ratio of the algorithm is tight up to additive terms.

4.3 Lower Bounds

In this section, we show that the competitive ratios obtained in Section 4.2 are optimal up to constants. We prove a lower bound for the online packing problem and another lower bound for the online covering problem.

Lemma 4.5. There is an instance of the online (fractional) packing problem with n constraints, such that for any B-competitive online algorithm, there exists a constraint for which

$$\sum_{j|i \in S(j)} y_j \geq c_i \frac{1}{B}\left(1 + \frac{\log_2 n}{2}\right) = c_i \Omega\left(\frac{\log n}{B}\right).$$

Proof. Consider the following instance with $n = 2^k$ packing constraints. The right-hand side of each packing constraint is 1. In the first iteration, a new variable $y(1,1)$ belonging to all constraints arrives. In the second iteration, two variables $y(2,1)$ and $y(2,2)$ arrive. Variable $y(2,1)$ belongs to the first 2^{k-1} constraints and $y(2,2)$ belongs to the last 2^{k-1} constraints. In the third iteration, four dual variables $y(3,1), y(3,2), y(3,3)$, and $y(3,4)$ arrive belonging each to 2^{k-2} packing constraints. The process ends in the $(k+1)$th iteration in which 2^k variables arrive, each belonging to a single packing constraint. The optimal solution sets the new 2^{i-1} variables in the ith iteration to 1. Since the algorithm is B-competitive we get the following set of constraints:

For each $1 \leq i \leq k+1$:

$$\sum_{j=1}^{i} \sum_{\ell=1}^{2^{j-1}} y(j,\ell) \geq \frac{2^{i-1}}{B}.$$

Multiplying the $(k+1)$th inequality by 1 and each inequality i, $1 \leq i \leq k$, by 2^{k-i} and summing up, we get that:

$$\sum_{j=1}^{k+1} 2^{k-j+1} \left(\sum_{\ell=1}^{2^{j-1}} y(j,\ell) \right) \geq \frac{1}{B}(k2^{k-1} + 2^k).$$

This follows since for each j, $\left(\sum_{\ell=1}^{2^{j-1}} y(j,\ell)\right)$ is multiplied by $1 + 1 + 2 + 4 + \cdots + 2^{k-j} = 2^{k-j+1}$. However, the left hand side is also the sum over all packing constraints. Thus, by an averaging argument, since there are 2^k constraints, we get that there exists a constraint whose right hand side is at least

$$\frac{1}{2^k \cdot B}\left(k2^{k-1} + 2^k\right) = \frac{1}{B}\left(1 + \frac{k}{2}\right).$$

Since $n = 2^k$ we get the desired bound. $\square$

Lemma 4.6. There is an instance of the online fractional covering problem with n variables such that any online algorithm is $\Omega(\log n)$-competitive on this instance.

Proof. Consider the following instance with $n = 2^k$ variables $x_1, x_2, \ldots, x_n$. The first constraint that arrives is $\sum_{i=1}^{n} x_i \geq 1$. If $\sum_{i=1}^{n/2} x_i \geq \sum_{i=n/2+1}^{n} x_i$, then the next constraint that is given is $\sum_{i=n/2+1}^{n} x_i \geq 1$, otherwise the constraint $\sum_{i=1}^{n/2} x_i \geq 1$ is given. The process of halving and then continuing with the smaller sum goes on until we remain with a single variable. The optimal offline solution satisfying all the constraints sets the last variable to one. However, for any online algorithm, the variables in each iteration that do not appear in subsequent iterations add up to at least $1/2$. There are $k + 1$ iterations, and thus the cost of any online algorithm is at least $1 + k/2$, concluding the proof. □

4.4 Two Warm-Up Problems

In this section, we demonstrate the use of the online primal–dual framework on two simple examples, a covering problem and a packing problem.

4.4.1 The Online Set-Cover Problem

Consider an online version of the offline set-cover problem discussed in Section 2.2. In the online version of the problem, a subset of the elements X arrive one-by-one in an online fashion. The algorithm has to cover each element upon arrival. The restriction is that sets already chosen to the cover by the online algorithm cannot be "unchosen."

This online setting exactly fits the online covering setting, since whenever a new element arrives a new constraint is added to the set-cover linear formulation. Hence, we can use any of the algorithms presented earlier in Section 4.2 to derive a monotonically increasing fractional solution to the set-cover problem.

Getting a randomized integral solution is extremely simple. We can simply adapt the randomized rounding procedure appearing in Section 2.2.2 to the online setting. Note that the algorithm picks *a priori* uniformly in random a threshold $\Theta(s) \in [0,1]$ for each set s. The algorithm then chooses the set s to the cover if $x_s \geq \Theta(s)$. Since x_s is monotonically increasing, the online algorithm simply chooses the set s to the cover once x_s reaches $\Theta(s)$. Note that the number of elements is not known in advance; however, we do choose $\Theta(s)$ by taking the minimum between $O(\log n)$ random variables. Thus, we can increase the number of random variables as the number of elements increases. The threshold $\Theta(s)$ can only decrease by doing that. The analysis is then straightforward proving that the algorithm produces a solution covering all requested elements with high probability, and its expected cost is $O(\log n)$ times the fractional solution. Since the fractional solution is

$O(\log m)$-competitive with respect to the optimal solution, we get that the integral algorithm is $O(\log n \log m)$-competitive.

In Section 5, we show how to obtain a deterministic online algorithm for the set cover problem with the same competitive ratio.

4.4.2 Online Routing

In this section, we give a simple example of an online packing problem. We study the problem of maximizing the throughput of scheduled virtual circuits. In the simplest version of the problem, we are given a graph $G = (V, E)$ with capacities $u(e)$ on the edges. A set of requests $r_i = (s_i, t_i)$ $(1 \le i \le n)$ arrives in an online fashion. To serve a request, the algorithm chooses a path between s_i and t_i and allocates a bandwidth of one unit on this path. The decisions of the algorithm are irrevocable, and all requests are permanent, meaning that once accepted they stay forever. If the total capacity routed on edge e is $\ell \cdot u(e)$, we say that the *load* on edge e is ℓ. Ideally, the total bandwidth allocated on any edge should not exceed its capacity (load $\ell \le 1$). The total profit of the algorithm is the number of requests served and as usual the competitive factor is defined with respect to the maximum number of requests that could have been served offline.

In the fractional version of the problem, the allocation is not restricted to an integral bandwidth equal to either 0 or 1; instead, we can allocate to each request a fractional bandwidth in the range $[0, 1]$. In addition, the bandwidth allocated to a request can be divided between several paths. This problem is an online version of the maximum multi-commodity flow problem. We describe the problem as a packing problem in Figure 4.2. For $r_i = (s_i, t_i)$, let $\mathbb{P}(r_i)$ be the set of

Primal	Dual
Minimize: $\sum_{e \in E} u(e)x(e) + \sum_{r_i} z(r_i)$	Maximize: $\sum_{r_i} \sum_{P \in \mathbb{P}(r_i)} f(r_i, P)$
subject to:	subject to:
$\forall r_i \in \mathbb{R}, P \in \mathbb{P}(r_i)$: $\sum_{e \in P} x(e) + z(r_i) \ge 1$	$\forall r_i \in \mathbb{R}$: $\sum_{P \in \mathbb{P}(r_i)} f(r_i, P) \le 1$
	$\forall e \in E$: $\sum_{r_i \in \mathbb{R}, P \in \mathbb{P}(r_i) \mid e \in P} f(r_i, P) \le u(e)$
$\forall r_i, z(r_i) \ge 0$, $\forall e, x(e) \ge 0$	$\forall r_i, P : f(r_i, P) \ge 0$

Fig. 4.2 The splittable routing problem (maximization) and its corresponding primal problem.

simple paths between s_i and t_i. For each $P \in \mathbb{P}(r_i)$, the variable $f(r_i, P)$ corresponds to the amount of flow (service) given to request r_i on the path P. The first set of constraints guarantees that each request gets at most a fractional flow (bandwidth) of 1. The second set of constraints follows from the capacity constraints on the edges. In the primal problem, we assign a variable $z(r_i)$ to each request r_i and a variable $x(e)$ to each edge in the graph.

This online setting exactly fits our online packing setting, as new dual variables arrive whenever a new request arrives. However, in each iteration it may happen that an exponential number of variables arrives. We show in the following that we can still overcome this problem and get an efficient algorithm. We present two algorithms for the problem, each having different properties. Let $d \le n$ be the length of the longest simple path between any two vertices in the graph. The first algorithm is the following:

```
Routing algorithm 1:
When a new request $r_i = (s_i, t_i)$ arrives:

  (1) if there exists a path $P \in \mathbb{P}(r_i)$ such that $\sum_{e \in P} x(e) < 1$:
      (a) Route the request on $P$ and set $f(r_i, P) \leftarrow 1$.
      (b) Set $z(r_i) \leftarrow 1$.
      (c) For each $e \in P$: $x(e) \leftarrow x(e)(1 + 1/u(e)) + 1/(|P| \cdot u(e))$, where $|P|$ is the length of the path $P$.
```

Theorem 4.7. The algorithm is 3-competitive and it violates the capacity of each edge by at most a factor of $O(\log d)$ (i.e., the load on each edge is at most $O(\log d)$).

The proof of Theorem 4.7 is almost identical to the proof of Theorem 4.1, and we leave it as a simple exercise to the reader. The main observation is that the exponential number of dual variables is not obstacle, since the algorithm only needs to check the validity of the condition in line (1). If, for example, $\mathbb{P}(r_i)$ is the set of all simple paths between s_i and t_i, then this condition can be easily checked by computing a shortest path between s_i and t_i.

The above algorithm may exceed the capacity of the edges. We next show a different algorithm that fully respects the capacities of the edges.

```
Routing algorithm 2:
Initially: $x(e) \leftarrow 0$.
When a new request $r_i = (s_i, t_i, \mathbb{P}(r_i))$ arrives:
    (1) If there exists a path $P(r_i) \in \mathbb{P}(r_i)$ of length $< 1$ with respect to $x(e)$:
        (a) Route the request on "any" path $P \in \mathbb{P}(r_i)$ with length $< 1$.
        (b) $z(r_i) \leftarrow 1$.
        (c) For each edge $e$ in $P(r_i)$:
    $x(e) \leftarrow x(e) \exp\left(\frac{\ln(1+n)}{u(e)}\right) + \frac{1}{n}\left[\exp\left(\frac{\ln(1+n)}{u(e)}\right) - 1\right]$.
```

Theorem 4.8. The algorithm is $O(u(\min)[\exp(\ln(1+n)/u(\min)) - 1])$-competitive and does not violate the capacity constraints. If $u(\min) \geq \log n$ then the algorithm is $O(\log n)$-competitive.

Proof. Note first that the function $(u(e)[\exp(\ln(1+n)/u(e)) - 1])$ is monotonically decreasing with respect to $u(e)$. Thus, when a request r_i is routed, the increase of the primal cost is at most:

$$1 + \sum_{e \in P} u(e) \left(x(e) \left[\exp\left(\frac{\ln(1+n)}{u(e)}\right) - 1 \right] + \frac{1}{n} \left[\exp\left(\frac{\ln(1+n)}{u(e)}\right) - 1 \right] \right).$$

This expression is at most

$$2\left(u(\min) \left[\exp\left(\frac{\ln(1+n)}{u(\min)}\right) - 1 \right] \right) + 1. \tag{4.10}$$

This follows since $z(r_i)$ is set to 1, and edges on the path P satisfy $\sum_{e \in P} x(e) \leq 1$. Each time a request is routed, the dual profit is 1.

Thus, the ratio between the primal and dual solutions is at most Expression (4.10).

Second, observe that the algorithm maintains a feasible primal solution at all times. This follows since $z(r_i)$ is set to 1 for any request for which the distance between s_i and t_i (with respect to the $x(e)$-variables) is strictly less than 1.

Finally, it remains to prove that the algorithm routes at most $u(e)$ requests on each edge e, and so the dual solution it maintains is feasible. To this end, observe that for each edge e, the value $x(e)$ is the sum of a geometric sequence with initial value $1/n[\exp(\ln(1+n)/u(e)) - 1]$ and a multiplier $q = \exp(\ln(1+n)/u(e))$. Thus, after $u(e)$ requests are routed through edge e, the value of $x(e)$ is

$$\begin{aligned} x(e) &= \frac{1}{n}\left(\exp\left(\frac{\ln(1+n)}{u(e)}\right) - 1\right)\frac{\exp\left(\frac{u(e)\ln(1+n)}{u(e)}\right) - 1}{\exp\left(\frac{\ln(1+n)}{u(e)}\right) - 1} \\ &= \frac{1}{n}(1+n-1) = 1. \end{aligned}$$

Since the algorithm never routes requests on edges for which $x(e) \geq 1$, we are done.

Finally, it is not hard to verify that when $u(\min) \geq \log n$, then

$$2\left(u(\min)\left[\exp\left(\frac{\ln(1+n)}{u(\min)}\right) - 1\right]\right) + 1 = O(\log n).$$

□

4.5 Notes

The definitions of the online covering/packing framework along with the basic algorithms in Section 4.2 and the lower bounds in Section 4.3 are based on the work of Buchbinder and Naor [32]. These algorithms draw on ideas from previous algorithms by Alon et al. [3, 4]. The third basic algorithm that incorporates the complementary slackness conditions into the online algorithm is based on the later work of Bansal et al. [14]. The online set-cover problem was considered in [3]. There, they gave a deterministic algorithm for the problem that is discussed later on in Section 5. The second routing algorithm in Section 4.4.2

appeared in [34]. It is basically an alternative description and analysis of a previous algorithm by Awerbuch et al. [11].

There is a long line of work on generating a near-optimal fractional solution for *offline* covering and packing problems, e.g. [52, 53, 54, 55, 75, 79, 84, 93]. All these methods take advantage of the offline nature of the problems. As several of these methods use primal–dual analysis, our approach can be viewed in a sense as an adaptation of these methods to the context of online computation.

5

The Online Set-Cover Problem

In Section 4, we saw how to derive a simple randomized $O(\log m \log n)$-competitive algorithm for the online set-cover problem. An intriguing question is whether we can obtain a deterministic algorithm for the problem with no degradation in the competitive ratio. A common approach for obtaining deterministic algorithms is *derandomization*, which means converting randomized algorithms into deterministic algorithms. For more details on derandomization methods we refer the reader to [6]. We note that one of the fundamental approaches to (offline) derandomization is the so-called method of conditional expectations or pessimistic estimators [6], which performs a deterministic rounding process. Coming up with a function that guides this process (the pessimistic estimator) is the key ingredient to a successful application of the method of conditional expectations.

Fractional solutions can often be converted into randomized algorithms, but it is usually much harder to perform this conversion online. Indeed, this conversion is possible for the online set-cover problem because of the way the fractional solution evolves in time. Furthermore, the randomized algorithm can be converted into a deterministic one by an *implicit* use of the method of conditional expectations, which leads to the development of a potential function guiding the online algorithm. The competitive factor of the deterministic algorithm obtained remains $O(\log m \log n)$.

5.1 Obtaining a Deterministic Algorithm

If the set system is not known in advance to the algorithm, then it is quite easy to verify that any deterministic algorithm is $\Omega(n)$-competitive. Thus, we assume that the universe of elements X is known to the algorithm along with the family of sets $\mathbb{S}$. It is unknown, however, which subset $X' \subseteq X$ of the elements the algorithm would eventually have to cover (as well as their order of appearance). Let $c(\mathbb{C}_{\mathrm{OPT}})$ denote the cost of the optimal solution. We design an online algorithm that is given a value $\alpha \geq c(\mathbb{C}_{\mathrm{OPT}})$ as input, and produces a feasible solution with cost $O(\alpha \log m \log n)$. However, in case the algorithm is given an infeasible value of α (i.e., $\alpha < c(\mathbb{C}_{\mathrm{OPT}})$), it may fail.

Our algorithm guesses the value of α by doubling. We start by guessing $\alpha = \min_{s\in\mathbb{S}} c_s$, and then run the algorithm with this value. If the algorithm fails we “forget” about all sets chosen so far to $\mathbb{C}$, update the value of α by doubling it, and continue on. We note that the cost of the sets that we have “forgotten” about can increase the cost of our solution by at most a factor of 2, since the value of α doubles in each step. In the final iteration, the value of α can be at most $2c(\mathbb{C}_{\mathrm{OPT}})$, losing altogether a factor of 4 as a result of the doubling procedure. Also, for each choice of α, our algorithm ignores all sets of cost exceeding α, and chooses all sets of cost at most α/m to $\mathbb{C}$.

The algorithm we design uses as a subroutine an online algorithm that generates an $O(\log m)$-competitive fractional solution, e.g., the online fractional algorithm presented in Section 4. This algorithm maintains a monotonically increasing fraction w_s for each set s. Let $w_e = \sum_{s|e\in s} w_s$. Note that the fractional algorithm guarantees that $w_e \geq 1$ for each element e which is requested. Initially, our algorithm starts with the empty cover $\mathbb{C} = \emptyset$. Define C to be the union of all the elements covered by members of $\mathbb{C}$. The following potential function is used throughout the algorithm:

$$\Phi = \sum_{e \notin C} n^{2w_e} + n \cdot \exp\left(\frac{1}{2\alpha} \sum_{s \in \mathbb{S}} (c_s \chi_{\mathbb{C}}(s) - 3 w_s c_s \log n)\right).$$

The function $\chi_{\mathbb{C}}$ above is the characteristic function of $\mathbb{C}$, that is, $\chi_{\mathbb{C}}(s) = 1$ if $s \in \mathbb{C}$, and $\chi_{\mathbb{C}}(s) = 0$ otherwise.

The deterministic online algorithm is as follows:

```
Run the algorithm presented in Section 4.2 to produce a mono-
tonically increasing online fractional solution. When the weight of
set $s$ is increased:

(1) If $s \in \mathbb{C}$ do nothing; Otherwise:
(2) $\Phi_{\text{start}}$ — value of $\Phi$ before increasing the weight of $s$.
(3) $\Phi_{\text{end}}$ — value of $\Phi$ after increasing the weight of $s$.
(4) $\Phi'_{\text{end}}$ — value of $\Phi$ after increasing the weight of $s$ and
    choosing $s$ to the cover.
(5) Choose set $s$ to $\mathbb{C}$ if $\Phi'_{\text{end}} \leq \Phi_{\text{start}}$.
(6) If $\Phi_{\text{start}} > \max\{\Phi'_{\text{end}}, \Phi_{\text{end}}\}$, return "FAIL."
```

In the following, we analyze the performance of the algorithm in a single iteration in which the condition $\alpha \geq c(\mathbb{C}_{\text{OPT}})$ is satisfied. We prove that the algorithm never fails in this iteration.

Lemma 5.1. Consider a step in which the weight of a set s is augmented by the algorithm. Let Φ_{start} and Φ_{end} be the values of the potential function Φ before and after the step, respectively. Then $\Phi_{\text{end}} \leq \Phi_{\text{start}}$. In particular, when $\alpha \geq c(\mathbb{C}_{\text{OPT}})$ the algorithm never fails.

Proof. Consider first the case in which $s \in \mathbb{C}$. In this case, the first term of the potential function remains unchanged. The second term of the potential function decreases and therefore the lemma holds.

The second case is when $s \notin \mathbb{C}$. The proof for this case is probabilistic. We prove that either including s in $\mathbb{C}$, or not including it, does not increase the potential function. Let w_s and $w_s + \delta_s$ denote the weight of s before and after the increase, respectively. Add set s to $\mathbb{C}$ with probability $1 - n^{-2\delta_s}$. (This is roughly equivalent to choosing s with probability $\delta_s/2$ and repeating it $4 \log n$ times.)

We first bound the expected value of the first term of the potential function. This is similar to the unweighted case. Consider an element $e \in X$ such that $e \notin C$. If $e \notin s$ then the term that corresponds to this element remains unchanged. Otherwise, let the weight of e before and after the step be w_e and $w_e + \delta_s$, respectively. Before the step, element e contributes to the first term of the potential function the value n^{2w_e}. The probability that we do not choose set s containing element e is $n^{-2\delta_s}$. Therefore, the expected contribution of element e to the potential function after the step is at most $n^{-2\delta_s} n^{2(w_e+\delta_s)} = n^{2w_e}$. By linearity of expectation it follows that the expected value of $\sum_{e \notin \mathbb{C}} n^{2w_e}$ after the step is precisely its value before the step.

It remains to bound the expected value of the second term of the potential function. Let

$$T = n \cdot \exp\left(\frac{1}{2\alpha} \sum_{s \in \mathbb{S}} (c_s \chi_{\mathbb{C}}(s) - 3 w_s c_s \log n)\right)$$

denote the value of the second term of the potential function before the step, and let T' denote the same term after the weight increase and the probabilistic choice of set s to the cover. Recall that $s \notin \mathbb{C}$. Then,

$$\mathbf{Exp}[T'] = T \cdot \exp\left(-\frac{1}{2\alpha} 3\delta_s c_s \log n\right) \cdot \mathbf{Exp}\left[\exp\left(\frac{1}{2\alpha} c_s \chi_{\mathbb{C}'}(s)\right)\right] \tag{5.1}$$

where $\chi_{\mathbb{C}'}(s) = 1$ is the indicator of the event that set s is chosen to the cover, which happens with probability $1 - n^{-2\delta_s}$. We bound the right-hand side of (5.1) as follows:

$$\begin{aligned}\mathbf{Exp}\left[\exp\left(\frac{1}{2\alpha} c_s \chi_{\mathbb{C}'}(s)\right)\right] &= n^{-2\delta_s} + (1 - n^{-2\delta_s}) \cdot \exp\left(\frac{c_s}{2\alpha}\right) && (5.2)\\ &\le 1 - 2\delta_s \log n + 2\delta_s \log n \exp\left(\frac{c_s}{2\alpha}\right) && (5.3)\end{aligned}$$

$$= 1 + 2\delta_s \log n \left(\exp\left(\frac{c_s}{2\alpha}\right) - 1\right) \tag{5.4}$$

$$\leq 1 + 2\delta_s \log n \frac{3c_s}{4\alpha} \tag{5.5}$$

$$\leq \exp\left(\frac{3\delta_s c_s \log n}{2\alpha}\right). \tag{5.6}$$

Here, (5.3) follows since for all $x \geq 0$ and $z \geq 1$, $e^{-x} + (1 - e^{-x}) \cdot z \leq 1 - x + x \cdot z$, (5.5) follows[1] since $e^y - 1 \leq 3y/2$ for all $0 \leq y \leq 1/2$, and (5.6) follows since $1 + x \leq e^x$ for all $x \geq 0$. Plugging in (5.1), we conclude that the expected value of the second term after the increase step and the probabilistic choices is at most

$$\mathbf{Exp}[T'] = T \cdot \exp\left(-\frac{1}{2\alpha} 3\delta_s c_s \log n\right) \cdot \exp\left(\frac{1}{2\alpha} 3\delta_s c_s \log n\right) \leq T.$$

By linearity of expectation it now follows that $\mathbf{Exp}[\Phi_{\text{end}}] \leq \Phi_{\text{start}}$. Therefore, either the event of choosing s to the cover, or the event of not choosing s to the cover, does not increase the potential function. We conclude that after each step $\Phi_{\text{end}} \leq \Phi_{\text{start}}$. □

Theorem 5.2. Throughout the algorithm, the following holds:

(i) Every $e \in X$ of weight $w_e \geq 1$ is covered, that is, $e \in C$.
(ii) $\sum_{s \in \mathbb{C}} c_s = \alpha \cdot O(\log m \log n)$.

Proof. Initially, the value of the potential function Φ is at most $n \cdot n^0 + n < n^2$, and hence it remains smaller than n^2 throughout the whole algorithm. Therefore, if for element e, $w_e \geq 1$, at some point of time, then $e \in C$, since otherwise the contribution of the term n^{2w_e} itself would be at least n^2. This proves part (i). To prove part (ii), note that by the same argument, throughout the algorithm

$$n \cdot \exp\left(\frac{1}{2\alpha} \sum_{s \in \mathbb{S}} c_s \chi_{\mathbb{C}}(s) - 3w_s c_s \log n\right) < n^2.$$

[1] Note that here we use the fact that $\alpha \geq c(\mathbb{C}_{\text{OPT}})$, since we ignored all sets with cost greater than α.

Therefore,

$$\sum_{s \in \mathbb{S}} c_s \chi_{\mathbb{C}}(s) \leq \sum_{s \in \mathbb{S}} 3 w_s c_s \log n + 2\alpha \log n,$$

and the desired result follows from the fact that the fractional solution is $O(\log m)$-competitive. □

5.2 Notes

The results in this section are based on the work of Alon et al. [3]. We note that the algorithms of [3] were not originally stated as primal–dual algorithms, yet interpreting them as primal–dual algorithms was the starting point of extending the primal–dual method to the realm of online computation. Alon et al. [3] also proved that any deterministic algorithm for the online set-cover problem is $\Omega(\log n \log m / (\log \log m + \log \log n))$-competititive for many values of m and n. In the unweighted version of the set-cover problem all sets are of unit cost and so the goal is to minimize the number of sets needed for covering the elements. Buchbinder and Naor [32] used the improved offline rounding technique of [89] to obtain an $O(\log d \log (n/\mathrm{OPT}))$-competitive algorithm, where d is the maximum frequency of an element (i.e., the maximum number of sets an element belongs to), n is the number of elements and OPT is the optimal size of the set cover.

Derandomization has proved to be useful for other online problems as well. Buchbinder and Naor [32], using derandomization methods, obtained an alternative routing algorithm that achieves the same competitiveness as the second routing algorithm in Section 4.4.2. The algorithm is based on the basic algorithms of Section 4.2 along with an online derandomization of the rounding method in [85, 86]. Another example of a derandomization of an online algorithm is by Buchbinder et al. [30] for the ad-auctions problem (see also Section 10). However, we note that we do not yet fully understand when derandomization methods can be applied in online settings. For example, for the online group Steiner problem on trees discussed in Section 11, there is currently only a randomized online algorithm, even though the offline randomized algorithm can be derandomized.

Another online variation of the set cover problem was considered in [10]. There, we are also given m sets and n elements that arrive one at a time. However, the goal of the online algorithm is to pick k sets so as to maximize the number of elements that are covered. The algorithm only gets credit for elements that are contained in a set that it selected *before* or *during* the step in which the element arrived. The authors of [10] showed a *randomized* $\Theta(\log m \log(n/k))$ competitive algorithm for the problem, where the bound is optimal for many values of n, m, and k. A different extension of the online set cover problem is studied in [5]. They considered an admission control problem where the goal is to minimize the number of rejections. The problem is solved by reducing it to an instance of online set cover with repetitions, i.e., each element may need to be covered several times.