A Survey of Recent Work on Designing Competitive Online Algorithms using the Primal-Dual Method

6.854 Advanced Algorithms Course Project

Fredrik Kjolstad
fredrikk@mit.edu

Ludwig Schmidt
ludwigs@mit.edu

Christos Tzamos
tzamos@mit.edu

Abstract

Online algorithms have become increasingly popular in the last few decades. One important reason is that they capture the uncertainty we face in many important domains, ranging from computational finance to internet security and catastrophe management. The primal-dual method is an approach that has been gaining popularity as a technique to arrive at approximations for NP-hard problems. More recently it has also been applied as a general framework to solve many online problems. In 2009, Buchbinder and Naor published a survey of applications of the primal-dual method to online algorithms. Since then new complex online problems have been tackled using this technique, such as the online node-weighted steiner tree problem, the k-server problem and online job-migration. We present a survey of recent applications of the primal-dual method to online problems.

1 Introduction

The primal-dual technique was successfully applied to approximate NP-Hard problems in the 1990's [11]. In the early 2000's an application of this technique to online algorithms was discovered, and has since then been used to solve many online problems [1].

The primal-dual method provides us with a surprisingly general framework to turn linear programs solving *offline* problems into approximate solvers for the *online* counter-parts. In online problems information is revealed in pieces, and we must make irrevocable decisions at each step based on the limited knowledge we have thus far. Correspondingly, the primal-dual method allows us to add one or more constraints to our linear program at a time, and it approximates the solution at each step without changing any assignments to variables it made in the past. Thus, we do not have to start from scratch when designing an online algorithm that has a known offline linear program. We can instead start from the linear program, and use the ideas from the primal-dual literature to devise an online algorithm.

In 2009, Buchbinder and Naor published a survey of applications of the primal-dual method to online algorithms [8]. The survey gave a thorough description of the technique. It then went on to survey several papers showing how the primal-dual method can be applied to solve a number of important online problems. Examples include online set-cover, caching, routing and ad-auction revenue maximization.

Since then the primal-dual method has been applied to solve additional problems in an online setting. In this paper we survey three new application of the technique to very different problems. This demonstrates the power and applicability of the primal-dual approach to online problems.

In section 2 we offer an introduction to the primal-dual method using ski rental as a running example. In section 3 we survey Naor, Panigrahi and Singh's paper from 2011, which presents the first online algorithm for the node-weighted steiner tree problem [13]. Section 4 surveys Bansal, Buchbinder and Naor's 2010 paper, which demonstrates the power of the primal-dual approach by using it to explore promising approaches to the online k-server problem: the holy grail of online algorithms [5]. Finally, in section 5 we survey Buchbinder, Jain and Menache's 2011 paper and companion technical report, in which they design an algorithm for online job-migration between geographically distributed cloud data centers to reduce electricity costs [6, 7]. This paper demonstrates the flexibility of the primal-dual technique, by applying it to the design of algorithms for complex real-world problems that out-perform reasonable greedy heuristics both in theory and in practice.

2 The Primal-Dual Approach

In this section we present the primal-dual method. We first explain the required background in linear programming and duality. We then introduce the technique using the ski rental problem as an example. Finally, we discuss the general primal-dual framework so that it can be applied to other problems.

We assume the reader is familiar with the online algorithm model, competitive analysis, approximation algorithms, (integer) linear programming, randomized rounding and the concepts of weak and strong duality and will not review these concepts here. For a brief discussion of these topics we refer the reader to section 2 of Buchbinder and Naor's survey [8].

2.1 Preliminaries

In addition to the standard theorems and definitions about linear programs and duality we introduce *approximate* complementary slackness. First, we review the concept of complementary slackness. Consider the following primal/dual linear programs:

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

subject to :

$$\forall j: \quad \sum_{i=1}^{n} a_{ij} x_i \geq b_j$$

$$\forall i: \quad x_i \geq 0$$

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

subject to :

$$\forall i: \quad \sum_{j=1}^{m} a_{ij} y_j \leq c_i$$

$$\forall j: \quad y_j \geq 0$$

A feasible solution $x = (x_i, \ldots, x_n)$ to the primal and a feasible solution $y = (y_i, \ldots, y_m)$ to the dual satisfy the complementary slackness condition if and only if

- for each i we have either $x_i = 0$ or $\sum_j a_{ij} y_j = c_i$
- and for each j we have either $y_j = 0$ or $\sum_i a_{ij} x_i = b_j$.

So each variable is either equal to 0 or the corresponding constraint is tight. A pair of feasible solutions x and y is optimal if and only if it satisifies the complementary slackness condition.

Approximate slackness relaxes the tightness conditions above. As a result, we only get an approximation ratio between the primal and dual solutions satisfying the approximate complementary slackness condition. As we will see in the context of online algorithms, this approximation ratio corresponds to the competitive ratio. The reason is that we formulate an LP for the offline version of a problem which we then approximate with an online algorithm. So the ratio between the primal and the dual solution gives a bound on the competitive ratio.

Now we precisely state the approximate complementary slackness condition. Let x and y be feasible solutions to the primal and dual linear programs. First we define the primal and dual complementary slackness conditions.

- A feasible solution x satisfies the *primal* complementary slackness condition if for $\alpha \geq 1$ we have either $x_i > 0$ or $c_i/\alpha \leq \sum_j a_{ij} y_j \leq c_i$ for any i.
- Similarly, y satisifies the *dual* complementary slackness condition if for $\beta \geq 1$ we have either $y_i > 0$ or $b_j \leq \sum_i a_{ij} x_i \leq b_j \beta$ for any j.

If x and y satisify the primal and dual complementary slackness conditions respectively then

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

Note that for $\alpha = \beta = 1$ this gives the ordinary complementary slackness theorem. The proof follows directly from applying the primal and dual complementary slackness properties to $\sum_i c_i x_i$.

2.2 Ski Rental

In this section we informally present the primal-dual approach by applying it to the ski rental problem.

In the ski rental problem a skier will go skiing several times in his life. Every time he goes skiing he has to decide whether to rent a pair of skis or to buy a pair of skis that he can use for all subsequent ski trips. Renting costs \$1, while buying skis costs \$B. The goal of the skier is to spend the least amount of money. The ski rental problem is interesting because the skier does not know beforehand how many days he will ski in his life — after all he may break his leg tomorrow. We assume an adversarial model where fate will ensure the worst possible outcome no matter what the skier decides.

2.2.1 The Online Ski Rental Algorithm

Against a malicious adversary, the optimal deterministic strategy that minimizes the competitive ratio has been known for a long time, and requires the skier to rent for B days before buying. If the last day of skiing is before day B then this strategy is optimal. On the other hand, if the last day of skiing is after day B, then the strategy is at most two times as expensive as the optimal strategy, OPT, which is to buy skis on the first day of skiing. This strategy achieves a competitive ratio of 2 (it is 2-competitive), and is optimal for deterministic strategies.

However, a better competitive ratio of $\frac{e}{e-1}$ can be achieved using randomization against an oblivious adversary. We will provide an optimal randomized algorithm using the primal-dual approach, but first we will discuss a deterministic primal-dual algorithm that achieves 2-competitiveness.

2.2.2 LP Formulation

We begin by formulating an integer linear program that captures the solution to the offline ski rental problem. We use an indicator variable $x \in \{0, 1\}$ that represents whether we buy the skis and a variable $d_i \in \{0, 1\}$ for every day that indicates whether we rent skis at day i. The objective we want to minimize then is:

$$B \cdot x + \sum_{i=1}^{n} d_i$$

subject to the constraints that for each day i:

$$x + d_i \geq 1$$

The optimal solution in an offline version is either $x = 1$ and $d_i = 0$ for each day or $x = 0$ and $d_i = 1$ for each day, whichever is best. That is, the optimal solution is to buy at once or never buy at all. This of course requires prior knowledge of n which we do not have in an online setting.

In an online setting we know the objective beforehand, but we begin without any constraints. Every day, a new constraint appears and we have to re-optimize our solution. However, since we cannot change the past we cannot undo any previous decisions. This means that we can never decrease any variables set in the past.

2.2.3 Deterministic Algorithm

We now describe how we can incrementally solve the linear program using the primal-dual approach. Later we argue that this does not cause the solution to deviate too much from OPT. In order to apply the primal-dual approach, we first relax the integer linear program to a linear program. Note that the optimal solution in the relaxed version is exactly the same as before (the integrality gap is 0).

The primal-dual method approximates the solution to the linear program incrementally by updating the primal and dual simultaneously. The primal and dual are given below.

$$(P) : \min B \cdot x + \sum_{i=1}^{n} d_i$$

$$\begin{aligned} &\text{subject to :} \\ &\forall \text{ day } i : && x + d_i \geq 1 \\ & && x \geq 0, \forall i : d_i \geq 0 \end{aligned}$$

$$(D) : \max \sum_{i=1}^{n} y_i$$

$$\begin{aligned} &\text{subject to :} && \textstyle\sum_{i=1}^{n} y_i \leq B \\ &\forall \text{ day } i : && 0 \leq y_i \leq 1 \end{aligned}$$

For the online version, on the i-th day a new constraint ($x+d_i \geq 1$) arrives in the primal problem (updating the objective as necessary) and a new variable (y_i) arrives in the dual. With the new constraint the primal problem may become infeasible (this happens if $x < 1$), so we need to apply an update rule to decide which variable to increase. If it is still feasible we do not have to do anything since we have already covered the new constraint. Otherwise, we need to decide which variable, x or d_i, to increase. We look at the dual to make this decision. Since the dual is a maximization problem, we would like to increase the new variable y_i as much as possible. Thus we increase y_i until we hit a constraint in the dual problem. This constraint corresponds to a variable in the primal problem, either d_i or x. We set this variable equal to 1.

Notice that the algorithm corresponds to exactly the same strategy we described earlier. That is, the skier rents skis the first B days and then buys them.

The analysis of the ski rental problem is crucial for understanding the concepts of the primal-dual method. The key idea is that every time the algorithm makes a change in the primal LP, a change in the dual LP happens as well so that the ratio between their corresponding objective values remains bounded by some function. This function gives the competitive ratio of the algorithm.

For the previous algorithm, we note that the update rule always maintains a feasible solution for both the primal and the dual. In order to bound the ratio between their objective functions we use approximate complementary slackness. By the update rule we have that:

- Whenever $x > 0$, $\sum_{i=1}^{n} y_i = B$ (tight)
- Whenever $d_i > 0$, $y_i = 1$ (tight)
- Whenever $y_i > 0$, $1 \leq x + d_i \leq 2$ (approximately tight)

So by approximate complementary slackness with $\alpha = 1$ and $\beta = 2$ we get that the primal objective is at most 2 times the dual objective. Therefore, it is at most 2 times the optimal offline solution and thus it is 2-competitive.

Note how we use the dual constraints to decide which primal variables to update. The idea is to update the variables so that we maintain the complementary slackness condition, which then allows us to bound the competitive ratio.

2.2.4 Randomized Algorithm

We now derive a randomized algorithm for the ski rental problem. We first find a fractional online solution and then use the fractional values of the primal variables to obtain the probabilities of buying and renting the skis for each day. In the deterministic case we always maintained a feasible primal solution and whenever a new constraint appeared on day i we either set d_i or x to 1. For the fractional case we do not have to set x directly to 1 when we reach B days, but instead we can gradually increase x every day. So for every new day i, if $x < 1$ we use the update rule:

- $x \leftarrow x(1 + 1/B) + 1/(c \cdot B)$
- $d_i \leftarrow 1 - x$
- $y_i \leftarrow 1$

where c is a constant to be determined later.

We move on to the analysis part of the algorithm. Let $x^{(i)}$ be the value of x on the i-th day. We first note that under the update rule chosen, after k days $x^{(k)} = \sum_{i=0}^{k-1}(1+1/B)^i/(c \cdot B) = [(1+1/B)^k - 1]/c$. It is obvious that the primal solution is always feasible since at any time i, $x^{(i)} + d_i = 1$ and no previous constraints are violated by the increase in x. We choose c so that the dual is also feasible. We want y_i to be 1 for at most B days otherwise the dual constraint $\sum_{i=1}^{n} y_i \leq B$ will be violated. So we require that after B days $x = 1$, which means that $c = (1 + 1/B)^k - 1$.

Moreover, we want the ratio between the primal objective and the dual objective to be bounded by a constant. We use a proof by induction to show that it is bounded by $(1 + 1/c)$. Initially, the objective functions of the primal and the dual are both 0 so the statement holds. Assuming that it holds for k days, we now prove that it holds for $k + 1$ days. If $k \geq B$ it clearly holds since the two objectives are left unchanged. Otherwise, we see that x increases by $x^{(k)}/B + 1/(c \cdot B)$ while d_{k+1} becomes $1 - x^{(k+1)}$. So the primal objective increases by $x^{(k)} + 1/c + 1 - x^{(k+1)} \leq 1 + 1/c$ since $x^{(k)} \leq x^{(k+1)}$. On the other hand the dual objective increases by 1 unit so the statement holds.

Therefore we get an online fractional algorithm for the ski rental problem with competitive ratio $1 + 1/c \approx e/(e-1)$ for large values of B. We now convert this into a randomized algorithm, interpreting the fractional values as probabilities.

Our randomized decision strategy is as follows: At the beginning we pick a random value v in the interval $[0, 1]$ and we buy skis on the first day i such that $x^{(i)} \geq v$. The probability of buying skis in the first i days is exactly $x^{(i)}$, i.e. the online fractional solution. The probability of renting skis on the i-th day is exactly $1 - x^{(i)}$. If we go skiing for n days the expected cost of our randomized algorithm is therefore $B \cdot x^{(n)} + \sum_{i=1}^{n}(1 - x^{(i)})$ which is exactly the same as the objective of the fractional algorithm. Thus the randomized algorithm is also $e/(e-1)$-competitive.

2.3 General Approach

We can generalize the approach used for ski rental to a wide class of linear programs. This class contains linear programs of the following form:

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

$$\begin{array}{ll} \text{subject to :} & \\ \text{for } 1 \leq j \leq m : & \sum_{i \in S(j)} x_i \geq 1 \\ \text{for } 1 \leq i \leq n : & x_i \geq 0 \end{array}$$

where each $S(j)$ is an arbitrary subset of the variables. Such LPs are called *covering LPs*, because they model problems where all elements of a set have to be covered by a collection of other sets. Examples of this include ski rental, where all days have to be covered by renting or buying skis, vertex cover, edge cover, set cover, and the problems in section 3–5.

The dual of a covering LP is called a *packing LP* and has the following form:

$$(D) : \max \sum_{j=1}^{m} y_j$$

$$\begin{array}{ll} \text{subject to :} & \\ \text{for } 1 \leq i \leq n : & \sum_{j|i \in S(j)} y_j \leq c_i \\ \text{for } 1 \leq j \leq m : & y_j \geq 0 \end{array}$$

For instance, the matching problem is the packing dual of the vertex covering primal, and independent set is the packing dual of the edge cover primal.

As the reader might see this duality nicely captures ski rental. The ski rental primal is a covering problem where we must cover all days with rented or bought skis. The corresponding dual is a packing problem where we pack the cost of renting skis into the cost of buying skis.

In the online version, a new constraint appears at every time step in the primal LP and a new variable arrives in the dual. We are only allowed to increase the variables at every time step.

Algorithm 1 provides an online fractional solution for such problems. Using a similar analysis as in the ski-rental problem we can prove that the algorithm is $O(\log d)$-competitive, where $d = \max_j\{|S_j|\}$ [8]. Note that this algorithm does not depend on the dual variables as in the randomized ski rental version. The dual variables are only used in order to bound its competitiveness by comparing the primal and the dual objectives. Using this general algorithm we can develop online algorithms for many different optimization problems as we will see in later sections. This algorithm gives us a competitive online fractional solution. However, it is our responsibility to use the fractional values to design our response.

Algorithm 1 Update rule for the primal and dual variables. The update rule is applied when new primal constraints appear.

```
1: while ∑_{i∈S(j)} x_i < 1 do
2:     for all i ∈ S(j) do
3:         x_i ← x_i(1 + 1/c_i) + 1/(|S(j)| · c_i)
4:     y_j ← y_j + 1
```

Other primal-dual algorithms use different update rules to obtain similar competitive ratios [8].

3 Graph Optimization

In this section we apply the primal-dual framework to three online graph optimization problems. We examine fractional connectivity, non-metric facility location [2] and node-weighted Steiner tree [13]. The best known algorithms for the metric facility location and the Steiner-tree problems have $O(\frac{\log n}{\log \log n})$ [10] and $O(\log n)$ [12] competitive ratios respectively. We now explain how the framework allows us to obtain comparable results in generalizations of those problems where no results were previously known.

3.1 Fractional Connectivity

In the fractional connectivity problem there is an underlying undirected graph $G = (V, E)$ where each edge e has a non-negative cost c_e. At each time step, a new request (S_i, T_i) arrives where S_i and T_i are two non-empty disjoint subsets of V and the goal is to send a unit of flow from S_i to T_i. To do this we buy capacity on the edges. If we buy capacity x_e on an edge e we have to pay $x_e \cdot c_e$. In the online setting, once we buy capacity on an edge we can reuse it for all subsequent requests. Also, we can never decrease the capacity of an edge, but we may increase it later by paying the extra cost.

We now apply the framework to this problem. We first formulate an LP, where for each edge we keep a variable x_e indicating the capacity of the edges we own. We note that a unit flow can be sent from S_i to T_i if and only if every (S_i, T_i) cut with respect to x_e has value at least 1. Therefore, the primal LP we get is:

$$(P) : \min \sum_{e \in E} c_e x_e$$

$$\begin{array}{ll} \text{subject to :} & \\ \forall \text{ day } i, \text{ every } (S_i, T_i) \text{ cut } C : & \sum_{e \in C} x_e \geq 1 \\ & x_e \geq 0 \end{array}$$

To get an online solution for the LP we use the framework developed in the previous section.The difference is that now instead of one constraint we get an exponential number of constraints added per request. We deal with this by finding the constraint that corresponds to the minimum (S_i, T_i) cut at every time step, and increase the primal and the dual variables as mentioned in the previous section. So the modified version of algorithm 1 performs the following update for every request:

As mentioned in the previous section, the algorithm is $O(\log d)$-competitive, where $d = \max_j\{|S_j|\}$.

Algorithm 2 Update rule for the primal and dual variables. The rule is applied when a new request (S_i,T_i) appears.

```
1: while the max S_i → T_i flow is less than 1 do
2:     Find the minimum (S_i, T_i) cut C
3:     for all edges e ∈ C do
4:         x_e ← x_e(1 + 1/c_e) + 1/(|C| · c_e)
5:     y_{i,C} ← y_{i,C} + 1
6: Make each edge e have capacity x_e
```

Here each S_j corresponds to a cut in the graph so $O(\log d) = O(\log \text{MAX-CUT}) = O(\log n)$ where $n = |V|$, the number of nodes in the graph.

The integral version of the connectivity problem where we are only allowed to buy an entire edge and not fractions is of particular interest since it captures many graph optimization problems. By randomly rounding the fractional capacity values on every edge we can obtain an integral solution. As we will see next, the facility location problem is one example for this.

3.2 Facility Location

In the facility location problem, there is a set F of n possible facilities that we can build along with a building cost c_f for every $f \in F$. In this problem clients arrive one at a time and want to be connected to at least one of the built facilities. Every client i has cost $c_{i,f}$ for every facility f that he may be connected to. So, whenever a client appears, we have to decide whether to connect him to some previously built facility or to build a new facility and connect him to it. Again, since this is an online setting we cannot undo the fact that a facility has been built or that a client has been connected to a facility.

We now show how to reduce this problem to the integral connectivity problem. For every facility $f \in F$, we create a tree where the root corresponds to f. We connect the root to a single node with an edge of cost c_f. Then we connect this node to the leaves where each leaf corresponds to every client. The edges connecting the client nodes to the intermediate node have cost equal to $c_{i,f}$. Note that each tree has unbounded size since the number of clients may be arbitrarily large and that in the online setting we do not know the edge costs of a client before he appears. However, we never use a part of the tree that corresponds to some client that has not arrived yet.

Every time a new client appears we need to connect a node corresponding to him in some tree to the root of that tree by buying edges. If the edge connecting the root of the tree to the intermediate node has already been bought it means that the facility has been built and that we only need to pay for connecting the client to that facility (buy the edge to the intermediate node). Thus, in the context of the connectivity problem we can see that each client is associated with two sets (S_i, T_i) where S_i is the set of nodes corresponding to him in every tree and T_i are the roots of all trees.

Therefore we can use the fractional algorithm we developed previously to get a fractional solution to the problem. Since the cut every time is exactly equal to n where n is the number of trees/possible facility locations the competitive ratio is $O(\log n)$. We now show how we can use the fractional solution to obtain an online randomized algorithm for facility location.

Throughout the course of the algorithm, we maintain $2\lceil log(k+1)\rceil$ values for every tree when k requests have occurred so far. These values are drawn uniformly at random from $[0, 1]$. Before any requests occur no tree has any such value. Whenever a request happens we sample more values so that every tree has exactly $2\lceil log(k+1)\rceil$ values. In order to decide which edges to buy we compute the minimum value for every tree, and buy all edges in that tree with fractional capacity greater than this value. For a single value drawn randomly out of $[0, 1]$ edge e has probability x_e of being selected. By the union bound for $2\lceil log(k+1)\rceil$ values the edge has probability less than $2\lceil log(k+1)\rceil x_e$ of being selected. So in total the expected cost is at most $2\lceil log(k+1)\rceil = O(\log k)$ times the cost of the fractional solution. This means that it is at most $O(\log n \log k)$ times the optimal offline algorithm.

The resulting solution however may not be feasible since it may be the case that a client is not connected to any facility. This happens with low probability: $1/k^2$ [13]. We can therefore connect client i to some root via the cheapest path. This is a lower bound to the value of the optimal offline solution since OPT has to connect client i to some facility. So overall the additional expected cost added when the randomization fails to connect a client to a facility is $\sum_k OPT/k^2 = O(1)OPT$ which is negligible.

So the competitive ratio of the randomized algorithm is $O(\log n \log k)$ for a total of k requests.

3.3 Node-Weighted Steiner Tree

In the node-weighted Steiner tree problem, there is an underlying graph $G = (V, E)$ where both vertices and edges have costs (c_v and c_e) respectively. At all times, we maintain a network of connected nodes in the graph. Initially this network is empty. Every time a request arrives on a vertex of the graph and it has to be connected to the network via a path formed by nodes and edges that we buy. As before, we need to decide at every time step which nodes and edges to buy to minimize the cost of keeping all request points connected. Once we buy a node or an edge we can never undo this action.

To obtain an online algorithm for this problem we use the following lemma [13]:

Lemma 3.1. *For any set of nodes $S = \{s_1, ..., s_k\}$ and any tree $T \subseteq G$ that contains all nodes in S, there exist nodes $v_2, v_3, ..., v_k \in T$, not necessarily distinct, for which:*

$$\sum_{i=2}^{k}[c(P_i^{v_i}) - c_{v_i}] + \sum_{v \in \{v_2, v_3, ..., v_k\}} c_v \leq O(\log k)c(T)$$

where $c(P_i^{v_i})$ is the cost of the cheapest path that goes through v_i and connects node s_i to some previous node s_j with $j < i$ and $c(T)$ is the cost of the tree. Costs include both node and edge costs.

Intuitively, this lemma says that if we are careful not to double-count the cost of some nodes while possibly counting the cost of the edges and all other nodes multiple times, we do not lose more than a logarithmic factor.

It follows that minimizing the quantity $\sum_{i=2}^{k}[c(P_i) - c_{v_i}] + \sum_{v \in \{v_2, v_3, ..., v_k\}} c_v$ for some nodes $v_2, ..., v_k$ we get an $O(\log k)$ approximation for the node weighted Steiner tree problem. To minimize the quantity in an online way, we can formulate the problem as a fractional connectivity problem and then try to round the fractional values. However, since this is very similar to the facility location problem, we will formulate the problem as a non-metric facility location instance.

The facility set F is the same as the set V of nodes and the corresponding costs are $c_f = c_v$. We view every request i at some node s_i as a client wanting to be connected to some facility where the cost for each facility f is $c_{i,f} = c(P_i^f) - c_f$ as in the lemma. Connecting the client i to a facility f corresponds to connecting the node s_i to some previous request node s_j with $j < i$ using the shortest path through node f. Using the previously developed algorithm for non-metric facility location and losing the additional $O(\log k)$ factor by the lemma, we get an $O(\log^2 k \log n)$-competitive algorithm for the node-weighted Steiner tree (k is the number of requests and n is the number of nodes in the graph).

4 Towards the Randomized k-server Conjecture: a Primal-Dual Approach

Recently, the primal-dual technique has been used to explore new approaches to the k-server problem [5]. Towards this end, the authors extend the primal-dual technique to deal with a wider range of LPs. We first demonstrate the extended primal-dual technique with an example from [5]. Then we briefly give an overview of the steps involved in applying the primal-dual technique to the k-server problem.

4.1 Extended Primal-Dual Technique

One of the main limitations of the original primal-dual technique is the restriction to non-negative constraints and variables. Using the basic approach described in section 2.3, it is impossible to include constraints like $x_i \geq x_j$, which have to be converted to $x_i - x_j \geq 0$ in an LP (note the negative coefficient of x_j). This condition restricts the set of problems we can model and makes some LP formulations more complex. The main technical contribution of [5] is to overcome these limitations.

We illustrate the extended primal-dual framework with the weighted caching problem. In this problem, we have a cache of size k and a total of n pages, each with a weight w_i. At each time step t, we receive a request for a page p_t. If p_t is not in the cache, we have to evict one page from the cache and replace it with p_t. Fetching a page p into the cache costs w_p. The goal is to minimize the total cost of fetching pages.

We now present a randomized $O(\log k)$ competitive algorithm for the weighted caching problem. While a randomized $O(\log k)$ competitive algorithm was known before [5], the following algorithm using the extended primal-dual technique has a significantly simpler analysis. It is worth noting that the first randomized $O(\log k)$ competitive algorithm also uses the primal-dual technique [4].

First, we define the primal LP.

$$(P): \min \sum_{p=1}^{n}\sum_{t=1}^{T} w_p \cdot z_{p,t} + \sum_{t=1}^{T} \infty \cdot y_{p_t,t}$$

subject to :

$\forall t$ and $S \subseteq [n]$ with $|S| > k$: $\sum_{p\in S} y_{p,t} \geq |S| - k$

$\forall t, p$: $z_{p,t} \geq y_{p,t-1} - y_{p,t}$

$\forall t, p$: $z_{p,t}, y_{p,t} \geq 0$

The variable $y_{p,t}$ denotes the fraction of page p missing at time t. Since $z_{p,t} \geq y_{p,t-1} - y_{p,t}$, the variable $z_{p,t}$ denotes the fraction of page p fetched at time t. Hence the first term of the objective captures the total cost of fetching pages. The second term of the objective guarantees that the requested page is always fetched into the cache. The first constraint ensures that only a total of k pages are in the cache at any time.

The dual LP is

$$(D): \max \sum_{t=1}^{T}\sum_{S} (|S| - k) a_{S,t}$$

subject to :

$\forall t, p \neq p_t$: $\sum_{S:p\in S} a_{S,t} - b_{p,t+1} + b_{p,t} \leq 0$

$\forall t, p$: $b_{p,t} \leq w_p$

$\forall t, p$ and $|S| > k$: $a_{t,S}, b_{p,t} \geq 0$

We now sketch the corresponding algorithm for iteratively constructing a solution to the primal. The following relation between the primal and dual is maintained at all times:

$$y_{p,t} \leftarrow \frac{1}{k} \cdot \left(\exp\left(\frac{b_{p,t+1}}{w_p} \cdot \ln(1+k) \right) - 1 \right)$$

When a new request for page p_t arrives, we set all variables $y_{p,t}$ for $p \neq p_t$ to the previous value for time $t-1$. Moreover, we set $y_{p_t,t} = b_{p_t,t} = 0$ because page p_t has to be in the cache. This violates the first primal constraint. In order to evict pages, we now increase $b_{p,t+1}$ for all pages that are (partially) in the cache. Note that increasing $b_{p,t+1}$ decreases the corresponding $y_{p,t}$. Moreover, we increase $a_{S,t}$ at the same rate so that the first dual constraint remains satisfied. Here, S is the set of pages that are at least partially in the cache.

By construction, the algorithm above always maintains feasible primal and dual solutions. Moreover, we can show that the increase in the primal objective is at most $O(\log k)$ times the change in dual objective. This is done by considering the derivatives of the primal and dual objectives with respect to $a_{S,t}$ and $b_{p,t+1}$. Since the relative change in primal and dual is bounded by $O(\log k)$, this shows that the fractional solution is $O(\log k)$ competitive. Moreover, the fractional solution can be rounded to a randomized algorithm with constant overhead in the competitive ratio [4].

4.2 The k-Server Problem

The k-server problem is one of the most important problems in the field of online algorithms. The problem statement is as follows: We have k servers located at up to k points in a metric space (essentially a set with a well-behaved distance function) and must serve requests appearing one after another. Each request is a point in the metric space and we serve it by moving one of the k servers to the request. The goal is to minimizhee the total distance travelled by all servers.

It is conjectured that there is a deterministic k-competitive algorithm for the k-server problem. While the best algorithm for the general case is $2k-1$ competitive, there are k-competitive algorithms for some special metrics like trees. For the randomized case, the conjectured competitive ratio is $O(\log k)$ and there has been recent work showing an $\tilde{O}(log^3 n\ log^2 k)$ approximation ratio [3]. While the results in [3] do not use the primal-dual approach for online algorithms, some of the techniques are inspired by problems that were first introduced in the context of the primal-dual approach.

One key difficulty of the k-server problem on general metric spaces is the weak structure of the distances between points. The most promising approach for overcoming this obstacle is to embed general metrics into hierarchically separated trees (HSTs). The distortion resulting from this embedding would still be sufficient to provide a polylogarithmic competitive ratio for the randomized k-server problem. Moreover, HSTs allow the k-server problem to be split recursively into subproblems at each node. The advantage of this approach is that it leads to simpler subproblems since all children of a node in an HST have the same distance (a *uniform* metric space).

In [5], the authors solve a restricted version of this subproblem called the *Allocation-C problem*. It is defined as follows: We have a uniform metric space with n points and up to k servers. For each request t, the number of available servers is limited to $k(t) \leq k$. Each request is described by a cost vector $h^t = (h^t(0), \ldots, h^t(k))$ where $h^t(j)$ is the cost of

serving the request t with j servers. The cost vectors satisfy two properties:

- The costs are monotonic, i.e. $h^t(j) \geq h^t(j+1)$. Using more servers for a request cannot increase the cost of serving it.
- The cost vector is convex. This means that we have $h^t(j+1) - h^t(j+2) \leq h^t(j) - h^t(j+1)$. So the marginal improvement achieved by adding a new server to the request decreases with the number of servers already at the request.

When a request comes in, we can move an arbitrary number of available servers to the request before serving it. The total cost incurred by the algorithm is divided into two parts:

- *Move-Cost*, the cost of moving servers.
- *Hit-Cost*, the cost of serving the requests.

The main contribution of [5] to the randomized k-server conjecture is an online algorithm for Allocation-C with the following guarantees (OPT is the optimal offline cost):

- The service cost is at most $(1+\epsilon)$ OPT.
- The movement cost is at most $O(\log \frac{k}{\epsilon})$ OPT.

This is a relevant result because Cote et al. showed that an algorithm satisfying similar conditions for the more general *Allocation problem* would yield a poly-logarithmic algorithm for the randomized k-server problem [9].

4.3 The Caching with Costs Problem

Instead of using the primal-dual approach directly, the authors first reduce the Allocation-C problem to another online problem. In the *Caching with Costs* problem, we have a uniform metric on l points and up to k available servers. For each request t, the number of available servers is limited to $k(t) \leq k$. Each request is described by a cost vector $c^t = (c_1^t, \ldots, c_l^t)$. We can redistribute servers among the l points when a new request comes in. In contrast to the Allocation-C problem, we now incur a cost c_i^t for each point i with no server present. The total cost incurred by the algorithm is divided into the movement cost and the hit cost.

The authors show that Allocation-C and Caching with Costs can be reduced to each other in an online fashion. Moreover, Caching with Costs can be simplified so that at time t, only one location p_t has non-zero cost c_{p_t}. The primal LP for Caching with Costs is:

$$(P): \min \sum_{t=1}^{T} c_{p_t,t} \cdot y_{p_t,t} + \sum_{i=1}^{n} \sum_{t=1}^{T+1} z_{i,t}$$

$$\begin{aligned} &\text{subject to :} \\ &\forall t \text{ and } S \subseteq [l]: && \textstyle\sum_{i:i\in S} y_{i,t} \geq |S| - k(t) \\ &\forall t, i: && z_{i,t} \geq y_{i,t-1} - y_{i,t} \\ &\forall t, i: && z_{i,t}, y_{i,t} \geq 0 \end{aligned}$$

The LP is similar to the one for the weighted caching problem described earlier. The variable $y_{i,t}$ indicates whether a server is located at point i at time t. Since $z_{i,t} \geq y_{i,t-1} - y_{i,t}$, the variable $z_{i,t}$ denotes the cost for moving a server to point i at time t. So the first term of the objective captures the hit cost and the second term the movement cost. The first constraint guarantees that at most $k(t)$ servers are in use at time t.

The dual LP is:

$$(D): \max \sum_{t=1}^{T} \sum_{S} (|S| - k(t)) \alpha_{t,S} + \sum_{i|\text{no server at } i} \gamma_i$$

$$\begin{aligned} &\text{subject to :} \\ &\forall t, i: && \textstyle\sum_{S:i\in S} \alpha_{t,S} + \beta_{i,t} - \beta_{i,t+1} \leq c_{i,t} \\ &\forall t, i: && \gamma_i \geq \beta_{i,1} \\ &\forall t, i: && 0 \leq \beta_{i,t} \leq 1 \\ &\forall t \text{ and } S \subseteq [l]: && a_{t,S} \geq 0 \end{aligned}$$

The algorithm for Caching with Costs is similar to weighted paging. Again, we maintain a relation between the primal and dual:

$$y_{i,t} \leftarrow \frac{\epsilon}{1+k} \left(\exp\left(\ln\left(1 + \frac{1+k}{\epsilon}\right) \cdot \beta_{i,t+1} \right) - 1 \right)$$

By considering the derivatives of the primal and dual objective functions, the authors prove the desired bounds for Caching with Costs which carry through to Allocation-C.

5 Job-Migration

In this section we demonstrate the applicability of the primal-dual method to real-world problems. We survey a systems paper, which is primarily of interest because of its applicability to system design, and not necessarily because of its theoretical value.

Linear programs capture many important real world problems. Since the primal-dual method gives

us a tool to design online algorithms for many problems that can be expressed as linear programs, we expect it to be useful for many real-world applications. Here we give one example by discussing its applicability to a data center job-migration problem.

The goal is to move jobs to data centers at locations with temporarily cheaper power. This is highly topical as energy over the last ten years has come to dominate data center operation costs. Buchbinder, Jain and Menache developed an online algorithm based on the primal-dual method that takes both energy costs and bandwidth costs into account [6, 7]. The algorithm is $O(\log H_0)$-competitive, where H_0 is the total number of servers in all data center.

The algorithm using the primal-dual method turned out to be too computationally expensive for practical use. However, the authors used it as inspiration in the design of an efficient non-intuitive algorithm. The authors cannot prove a worst-case bound for this algorithm, but they show empirically that it outperforms two reasonable greedy heuristics on real-world data. In fact, it performs to within $4\% - 6\%$ of the optimum on this data.

5.1 Problem Formulation

Many data center operators have many data centers that are geographically distributed. Since electricity prices typically vary between geographical locations, migrating tasks to data centers with cheaper electricity has recently been proposed. Furthermore, energy prices fluctuate and the cheapest data center at one point in time may not be the cheapest later on. This motivates online algorithms that monitor electricity prices and at any time move tasks to data centers with the cheapest electricity. However, moving data is not free and typically incurs a cost proportional to the size of the moved data. These competing constraints make the problem non-trivial and greedy heuristics with provable performance are not feasible.

Specifically, the domain consists of n data centers, where each data center i contains H_i servers. H_0 denotes the total number of servers in all data centers. At any time there are B jobs in the system and each job must be assigned to some server[1]. At each time step t the energy costs of data centers change and we must decide whether to move jobs. It costs d_i to move data out of data center i, and the variable $z_{i,j,t}$ denotes the fractional number of jobs migrated from server j in data center i at time t. Finally, $c_{i,j,t}$ is the energy cost of operating server j in data center i at time t, and $y_{i,j,t}$ is the fraction of server j of data center i that is utilized at time t. Jobs are defined as using exactly one server, so the number of servers needed at data center i at time t is $\lceil \sum_{j=1}^{H_i} y_{i,j,t} \rceil$. Since the number of servers in each data center is expected to be large we ignore the added fraction introduced by the ceiling operator.

The following (primal) linear program captures the *offline* data center job-migration problem:

$$(P): \min \sum_{i=1}^{n} \sum_{j=1}^{H_i} \sum_{t=1}^{T} d_i \cdot z_{i,j,t} + \sum_{i=1}^{n} \sum_{j=1}^{H_i} \sum_{t=1}^{T} c_{i,j,t} \cdot y_{i,j,t}$$

subject to :

$$\begin{aligned}
\forall t: &\quad \textstyle\sum_{i=1}^{n} \sum_{j=1}^{H_i} y_{i,j,t} \geq B \\
\forall i,j,t: &\quad z_{i,j,t} \geq y_{i,j,t-1} - y_{i,j,t} \\
\forall i,j,t: &\quad y_{i,j,t} \leq 1 \\
\forall i,j,t: &\quad y_{i,j,t} \geq 0 \\
\forall i,j,t: &\quad z_{i,j,y} \geq 0
\end{aligned}$$

The objective function minimizes the sum of the bandwidth cost (first term) and the energy cost (second term). The first constraint ensures the allocation can service the workload B. The second constraint enforces that the amount of work leaving a server is at least as much as the work that the server lost (it may be more as a server may also receive work). The third constraint prevents servers from being oversubscribed, since each server can only execute one task. The fourth and fifth constraints prevent server and bandwidth usage from being negative.

The corresponding dual linear program is:

$$(D): \max \sum_{t=1}^{T} B \cdot a_t - \sum_{i=1}^{n} \sum_{j=1}^{H_i} \sum_{t=1}^{T} s_{i,j,t}$$

s.t. :

$$\begin{aligned}
\forall i,j,t: &\quad -c_{i,j,t} + a_t + b_{i,j,t} - b_{i,j,t+1} - s_{i,j,t} \leq 0 \\
\forall i,j,t: &\quad b_{i,j,t} \leq d_i \\
\forall i,j,t: &\quad b_{i,j,t} \geq 0 \\
\forall i,j,t: &\quad s_{i,j,t} \geq 0
\end{aligned}$$

5.2 Primal-Dual Online Algorithm

Given these linear programs we can devise an online algorithm based on the primal-dual method. That is, we simultaneously update the primal (P) and dual (D) in such a way that we maintain feasible solutions. We can then derive an upper bound for the operation cost of the dual, which by weak duality becomes the

[1]The technical report generalizes to variable workloads B_t.

lower bound of the primal, and hence a bound on the competitive ratio of the algorithm.

Algorithm 3 shows the update rule for the dual variables. The update rule is applied for each server (i, j), and the cost vector c_t is supplied as input. The cost vector c_t is defined to be an *elementary* cost vector. That is, every entry is 0 except for the cost of server i, j at that invocation of the update rule. Note that a more precise name would be c_{i_t,j_t}, but c_t is used instead for notational convenience. Note also that the reduction to elementary cost vectors does not affect the solution.

Algorithm 3 Procedure to update dual variables. The procedure is executed for every server (i, j).

Input: Data-center i_t, server j_t, cost c_t (at time t)

1: **while** $y_{i_t,j_t,t} \neq 0$ & $-c_t + a_t + b_{i_t,j_t,t} - b_{i_t,j_t,t+1} \neq 0$ **do**
2: Increase a_t continuously
3: **for all** $i, j \neq i_t, j_t$ such that $y_{i,j,t} \leq 1$ **do**
4: Increase $b_{i,j,t+1}$ with rate $\frac{d \cdot b_{i,j,t+1}}{d \cdot a_t} = 1$
5: **for all** $i, j \neq i_t, j_t$ such that $y_{i,j,t} = 1$ **do**
6: Increase $s_{i,j,t}$ with rate $\frac{d \cdot s_{i,j,t}}{d \cdot a_t} = 1$
7: **for all** $i, j = i_t, j_t$ **do**
8: Decrease $b_{i_t,j_t,t+1}$ so that $\sum_{i,j \neq i_t,j_t} \frac{d \cdot y_{i,j,t}}{d \cdot a_t} = -\frac{d \cdot y_{i_t,j_t,t}}{d \cdot a_t}$

The objective of the dual algorithm is to maximize the dual profit, and it is therefore beneficial to increase a_t as much as possible, without violating constraints. However, the update rule is expressed in terms of continuous updates to a_t. In practice, the algorithm would be implemented in terms of discrete time iterations. It is possible to achieve this by searching for a scalar value of a_t up to the required precision level.

Finally, as stated earlier, the primal variable, $y_{i,j,t}$ is updated together with the corresponding dual variable $b_{i_t,j_t,t+1}$. The following update rule is applied to the primal:

$$y_{i,j,t} \leftarrow \frac{1}{H_0}\left(\exp\left(\ln(1 + H_0)\frac{b_{i,j,t+1}}{d_i}\right) - 1\right)$$

Note that the process is driven by dual updates and that we determine termination based on these. The primal is tied to the dual by (weak) duality, and we can therefore update the primal variables based on their corresponding dual variable.

Theorem 5.1. *The online algorithm is $O(\log(H_0))$-competitive.*

We do not provide a proof here, but it is based on deriving a lower bound for the value of the primal minimization problem using the value of the dual maximization problem. This takes advantage of the fact that the primal and dual are linked through the primal update rule. A detailed proof is given in [6].

5.3 Efficient Online Algorithm

As mentioned at the beginning of the section, the primal-dual algorithm in section 5.2 is too computationally expensive for practical usage. Reasons for this include the cost of the discrete time steps in the dual update rule, and the cost of managing one elementary cost vector per dual variable. Because of this the designers of the primal-dual algorithm also designed a practical $O(n^2)$ algorithm. This algorithm is based on the primal-dual algorithm and uses the same ideas, but looses the provable competitive ratio in exchange for efficiency. To evaluate the algorithm the authors performed empirical experiments comparing the algorithm with reasonable, and much more intuitive, heuristics on real electricity price data.

The algorithm works as follows. For each iteration the update rule specifies how much work to move from data center i to data center k. A detailed description can be found in [7].

Algorithm 4 Efficient job migration algorithm

Input: The cost vector $c = (c_1, c_2, ..., c_n)$, and the current load vector $y = (y_1, y_2, ..., y_n)$

1: **for** $k \leftarrow 1...n$ **do**
2: **for** $i \leftarrow k + 1...n$ **do**
3: Move load from DC i to DC k according to the following migration rule

$$\min\left\{y_i, H_k - y_k, s_1 \cdot \frac{c_{i,k} y_i}{d_{i,k}} \cdot (y_k + s_2)\right\}$$

where $s_1, s_2 > 0$

To evaluate the algorithm the authors compared it to the following heuristics:

Move to Cheap At each time step, move all tasks to the servers with the least expensive electricity. This fully optimizes power usage, but does not optimize for bandwidth at all.

MimicOPT At each time step, attempt to mimic OPT by solving an LP with the electricity costs up to the current time t (in practice a time window). Note that this algorithm, as opposed to

the primal-dual inspired algorithm, makes no attempt to take future costs into account.

They ran a simulated experiments comparing the efficient primal-dual inspired online algorithm, Move to Cheap and MimicOPT to OPT. The experiments were performed on publicly available electricity pricing information from 30 US locations covering January 2006 through March 2009. The results show that the efficient primal-dual inspired algorithm performs to within 5.7% of OPT, while Move to Cheap and MimicOPT only performs to within 68.5% and 10.4% of OPT respectively. This clearly motivates using an algorithm that takes ideas from the algorithm developed using the primal-dual method for complex problems. Even though the provable competitive ratio is lost, such algorithms may perform better than intuitive heuristics since they more systematically take complex constraints into account.

6 Conclusions

In this survey we introduced the primal-dual method as applied to online problems, and surveyed three papers that apply it to complex problems. The first paper used the method to design the first algorithm to solve to the online node-weighted steiner tree problem. The second paper used the primal-dual method to make progress on a promising path towards the solution of the important k-server problem. Finally, the third paper demonstrated how the primal-dual method can be applied to design online algorithms for real-world problems that can be expressed as linear programs. We expect the technique to be applied to more problems in the future, and we are especially excited about its promise in systems design where online problems abound.

References

[1] N. Alon, B. Awerbuch, and Y. Azar. The online set cover problem. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, STOC, pages 100–105, 2003.

[2] N. Alon, B. Awerbuch, Y. Azar, N. Buchbinder, and J. S. Naor. A general approach to online network optimization problems. *ACM Trans. Algorithms*, 2, 2006.

[3] N. Bansal, N. Buchbinder, A. Madry, and J. S. Naor. A polylogarithmic-competitive algorithm for the k-server problem. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2011.

[4] N. Bansal, N. Buchbinder, and J. S. Naor. A primal-dual randomized algorithm for weighted paging. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2007.

[5] N. Bansal, N. Buchbinder, and J. S. Naor. Towards the randomized k-server conjecture: a primal–dual approach. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2010.

[6] N. Buchbinder, N. Jain, and I. Menache. Online job-migration for reducing the electricity bill in the cloud. In *International Conferences on Networking (NETWORKING)*, 2011.

[7] N. Buchbinder, N. Jain, and I. Menache. Online job-migration for reducing the electricity bill in the cloud. Technical report, Microsoft Research, 2011.

[8] N. Buchbinder and J. S. Naor. The design of competitive online algorithms via a primal–dual approach. *Foundations and Trends in Theoretical Computer Science*, 3(2–3), 2009.

[9] A. Coté, A. Meyerson, and L. Poplawski. Randomized k-server on hierarchical binary trees. In *ACM Symposium on Theory of Computing (STOC)*, 2008.

[10] D. Fotakis. On the competitive ratio for online facility location. In *Automata, Languages and Programming*, volume 2719 of *Lecture Notes in Computer Science*, pages 190–190. Springer Berlin / Heidelberg, 2003.

[11] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. In *Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, SODA, pages 307–316, 1992.

[12] M. Imase and B. M. Waxman. Dynamic steiner tree problem. *SIAM J. Discrete Math.*, 1991.

[13] J. S. Naor, D. Panigrahi, and M. Singh. Online node-weighted steiner tree and related problems. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2011.