

Lecture 21: Nov 10

Lecturer: Guanghui Wang

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

Last lecture, we discussed the UCB and its guarantees on performance. We saw that in addition it enjoys an $\mathcal{O}\left(\sum_{a \neq a^*} \frac{\log T}{\Delta_a}\right)$ style guarantee on pseudo-regret bound. It turns out that the regret bounds that we have investigated thus far, increase proportionally to the total number of arms K (either linearly or with a square-root dependence). Many practical applications of the limited-information feedback paradigm are very large-scale, and so the number of arms K can be either exceptionally large or even infinite in size. In this case, the guarantees that we have discussed in class so far become unsuitable. The central issue with algorithms like UCB is that they do not model structure across arms. In today's lecture, we will explore several examples of such large-scale bandit problems with structure across arms, and show that the ideas from classical multi-armed bandit algorithms can be adapted to these large-scale settings. We will also discuss popular applications of these large-scale bandit problems.

21.1. Example 1: Combinatorial structure

Suppose that we have moved to a new city, and we are commuting from home to work every morning¹. Then, we would want to pick the path that requires the least expected travel time; however, we are uncertain about the travel time along different routes and we need to explore to figure this shortest path out.

This constitutes an online-shortest-path problem on a graph, and the vertices represent intermediate stopping points (e.g. intersections, or stop signs). As Figure 21.1 demonstrates, the mean travel time along an edge e is denoted by θ_e , and a path $\mathbf{e} := (e_1, \dots, e_n)$ from source s to destination t will lead to total travel time $\sum_{i=1}^n \theta_{e_i}$. Our goal is to minimize pseudo-regret with respect to the best path, i.e. the one that minimizes the total travel time. We denote this path by $\mathbf{e}^* := (e_1^*, \dots, e_n^*)$.

This is, of course, an example of the multi-armed bandit problem (where the “arms” represent all possible paths in the graph). Unfortunately, at first glance it seems to be very large in scale: the number of total paths can scale **exponentially** in the number of vertices/edges, and since we have seen that pseudo-regret tends to increase with the number of arms in a linear fashion, this could lead to a very suboptimal guarantee. Moreover, a UCB-type algorithm in its naive form would also need to enumerate all possible paths, which is also computationally expensive for the same reason of exponential scaling.

1. and we have decided to experiment on our own instead of placing our trust in Google Maps.

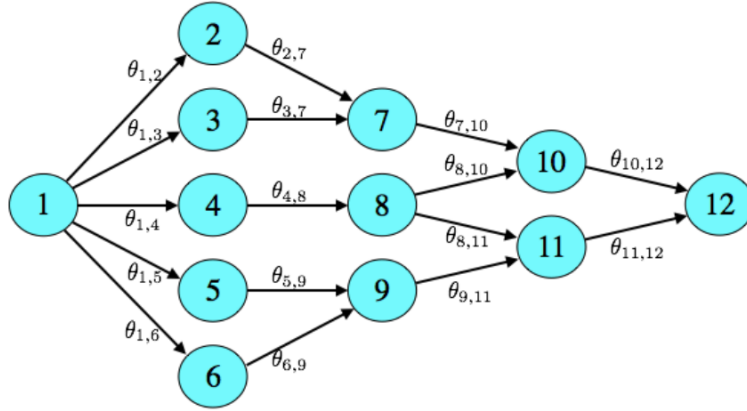


Figure 21.1: A depiction of the shortest-path problem with 12 vertices. Here, vertex 1 is the source vertex and 12 is the target (destination) vertex.

However, there is significant *structure* in this problem whereby we can do much better. The central idea that can be exploited is that at every round, we observe not only the total travel time, but also the travel time along each of the edge segments of the path that we tried. Moreover, several edges are common to multiple paths, so observing the travel time on a particular edge will tell us about the travel time of all paths that use that edge. It turns out that we can leverage several algorithms that leverage this *combinatorial structure*, which we formally define below.

Definition 1 A combinatorial bandit problem is one in which the action space is given by $\mathcal{A} \subset \{0,1\}^d$ for some dimension d , and the reward corresponding to any action $\mathbf{a} \in \mathcal{A}$ is given by

$$G_{t,\mathbf{a}} = \langle \mathbf{a}, \boldsymbol{\theta}^* \rangle + W_t, \quad (21.1)$$

where W_t denotes some random noise and $\boldsymbol{\theta}^* \in \mathbb{R}^d$ denotes an unknown parameter which we call the “reward vector”. The feedback received by the learner could involve only $G_{t,\mathbf{a}}$, but it could additionally involve observing the individual components θ_i^* (along with some noise) corresponding to the indices $i \in \{1, \dots, d\}$ for which $a_i = 1$. The latter, richer type of feedback is often called **semi-bandit feedback**.

The above example of finding the shortest path clearly is an instance of the combinatorial bandit problem specified in Definition 1. To see this, we observe that:

- The dimension of the problem is equal to the number of edges in the graph, i.e. $d = |E|$. Henceforth, we use i to index an edge.
- The components of the reward vector $\boldsymbol{\theta}^*$ represent the “negative” travel time $-\theta_i$, i.e. to maximize reward, we wish to minimize travel time.

- The action space represents all possible paths from the source vertex s to the destination vertex t . Only edges that are on the candidate path are marked with 1; all other edges are marked with 0, i.e. $a_i = 1$ if and only if the edge i is on the selected path. Moreover, the action set is constrained because the set of edges marked with 1 need to constitute a valid path; therefore, it is only a *subset* of $\{0, 1\}^{|E|}$.
- Finally, our observed feedback could involve the total travel time involved *or* the individual travel times along the edge segments of the route.

Note that the combinatorial bandit problem is an instance of the MAB problem with at most 2^d arms; however, naively applying an algorithm like UCB would then lead to regret that scales exponentially in the parameter d . In our shortest path example, this would scale exponentially in the number of edges—typically unacceptable for such applications! The central issue is that UCB in its naive form does not exploit the strong structure that is present across arms in this formulation. Figure 21.1 depicts that some edges are common to more than one path from source 1 to destination 12: for example, observing $\theta_{10,12}$ alone will yield partial information about at least 3 out of the 6 candidate paths! More generally, we notice that knowing the “reward vector” θ^* to a reasonable extent actually suffices to make decisions about *all* the candidate actions. We now explore a general formulation of such *parameterized* bandit problems, and describe at a high level algorithms that will leverage this structure.

21.2. A general formulation: Linearly parameterized bandits

The combinatorial bandit problem in Definition 1 is an instance of a *linearly parameterized* bandit problem, which we now define for completeness.

Definition 2 *The (stochastic) linearly parameterized bandit problem involves an action set \mathcal{A} that can be either discrete (as in the preceding example) or more generally continuous. The action set, even when continuous, is typically **bounded**, e.g. $\|\mathcal{A}\| \leq 1$ in some norm $\|\cdot\|$, and reward function*

$$G_{t,a} = \langle \mathbf{a}, \theta^* \rangle + W_t, \quad (21.2)$$

where W_t denotes some random noise and $\theta^* \in \mathbb{R}^d$ denotes an unknown parameter which we call the “reward vector”. In this model, we typically only observe the reward feedback $G_{t,a}$.

For all practical purposes, algorithms that are used for the combinatorial bandits problem (Definition 1) are very similar to algorithms that are used for the more general linearly parameterized bandit problem (Definition 2), and so we now focus on the latter which is a more general case (since \mathcal{A} can now be unbounded/continuous, and we only observe the reward feedback $G_{t,a}$). To think of where this more general case could arise, consider an example of network routing in which the parameters θ_e constitute congestion amounts, and the action set becomes continuous as different *fractions* of traffic could be routed on different paths.

In this linear bandit formulation, we denote the action selected at round t by \mathbf{A}_t . As before, pseudo-regret is defined with respect to the best arm $\mathbf{a}^* := \arg \max_{\mathbf{a} \in \mathcal{A}} \langle \mathbf{a}, \boldsymbol{\theta}^* \rangle$, i.e.

$$\bar{R}_T := \langle \mathbf{a}^*, \boldsymbol{\theta}^* \rangle - \mathbb{E} \left[\sum_{t=1}^T G_{t, \mathbf{A}_t} \right]. \quad (21.3)$$

Despite the action set being exponential or possibly infinite in size, it is in fact possible to design algorithms that achieve a pseudo-regret guarantee of the form $\bar{R}_T = \mathcal{O}(d\sqrt{T})$, i.e. *linear* in the dimension of the reward vector! In our preceding example of shortest path, this means that we would be able to find the shortest path in time that is *linear* rather than exponential in the number of edges in the graph.

These algorithms will directly use the strong structure in the problem, and are described next. We will not discuss the proof and approach to these in detail in this lecture.

21.2.1 The LinUCB algorithm

The first natural approach is to see whether the ideas in UCB could be generalized to this linearly parameterized bandit problem. We recall that UCB had three steps involved in it:

- Construct the sample means as estimates of the rewards of each arm.
- Construct confidence intervals on these estimates.
- Use the “optimism” principle to pick the *upper* confidence bound, and the action \mathbf{A}_t to be the one that maximizes this upper confidence bound.

We will now follow the above roadmap to generalize UCB to linearly parameterized bandits. The reason we start with the somewhat more mysterious “optimism” principle is because it turns out to be easier to generalize into an algorithm: we will see there-after that it can be connected to a different type of exploration-exploitation tradeoff here as well.

The *LinUCB* (or *OFUL*) algorithm generalizes UCB to linearly parameterized bandits in the following steps (described first at a high level):

- Step 1: Construct an estimate of the reward vector, which we denote by $\hat{\boldsymbol{\theta}}_t$.
- Step 2: Define a *confidence set* for this reward vector (that includes the sample mean $\hat{\boldsymbol{\theta}}_t$), which we denote by \mathcal{C}_t . The confidence set is a subset of the action set \mathcal{A} and is depicted in Figure 21.2. This generalizes the idea of a confidence interval.
- Step 3: Pick $\mathbf{A}_t = \arg \max_{\mathbf{a} \in \mathcal{A}} \max_{\boldsymbol{\theta} \in \mathcal{C}_t} \langle \mathbf{a}, \boldsymbol{\theta} \rangle$. This generalizes the optimism principle, as for a given action \mathbf{a} , we have chosen the parameter within the confidence set that is the “best-case” scenario for maximizing the reward under that action.

We now elaborate on Steps 1 and 2 below.

Constructing the estimate $\hat{\boldsymbol{\theta}}_t$: At round t , note that we have taken actions $\mathbf{A}_1, \dots, \mathbf{A}_{t-1}$ and observed rewards $G_{1, \mathbf{A}_1}, \dots, G_{t-1, \mathbf{A}_{t-1}}$. Moreover, for a round s we have $G_{s, \mathbf{A}_s} = \langle \mathbf{A}_s, \boldsymbol{\theta}^* \rangle + W_s$. Therefore, we can think of the actions $\{\mathbf{A}_s\}_{s=1}^{t-1}$ as *covariates*, and the observed rewards $\{G_{s, \mathbf{A}_s}\}_{s=1}^{t-1}$ as *responses* in a linear model with the unknown parameter

θ^* . Thus, it is natural to use a *least-squares* estimator to estimate $\hat{\theta}_t$ from the previous $t - 1$ collected samples, with an extra “ridge” parameter λ to avoid numerical issues in the inverse. This is denoted by

$$\hat{\theta}_t := (\mathbf{V}_{t-1} + \lambda \mathbf{I})^{-1} \sum_{s=1}^{t-1} \mathbf{A}_s G_{s, \mathbf{A}_s}, \quad (21.4)$$

where $\mathbf{V}_{t-1} := \sum_{s=1}^{t-1} \mathbf{A}_s \mathbf{A}_s^\top$ denotes the regression “Gram matrix”.

Exercise 1 *Verify for yourself that this squares up with your understanding of linear least-squares regression. Start by noting that you can write $\mathbf{V}_{t-1} = \mathbf{X}_{t-1}^\top \mathbf{X}_{t-1}$, where*

$$\mathbf{X}_{t-1} := \begin{bmatrix} \mathbf{A}_1^\top \\ \mathbf{A}_2^\top \\ \vdots \\ \mathbf{A}_{t-1}^\top \end{bmatrix}; \text{ then use the notes provided in Review 2.}$$

Constructing the confidence set \mathcal{C}_t : Thus, we have seen that insights from linear regression help us construct an estimate of the reward vector $\hat{\theta}_t$. However, this estimate may be unreliable, particularly along certain “directions”, depending on which actions have been sampled thus far. For example, the shortest path problem illustrates that observing certain paths would only inform us about the rewards on some of the edges, not others. To take a concrete example, the path that goes from $1 \rightarrow 2 \rightarrow 7 \rightarrow 10 \rightarrow 12$ gives us information only about the parameters $\theta_{1,2}, \theta_{2,7}, \theta_{7,10}, \theta_{10,12}$. This means that, just like in the simpler MAB problem, a *greedy* algorithm that attempts to maximize $\langle \hat{\theta}_t, \mathbf{a} \rangle$ may not always do well.

Like in regular UCB, the first step to remedy is to construct a *confidence set* that measures the uncertainty in the estimation of the parameter θ^* along each direction $i = 1, \dots, d$. As pictured in Figure 21.1, this turns out to take the nice geometric form of an *ellipsoid*; axes along which the radius is smaller denote directions i for which we have very good estimates of θ_i^* , but axes along which the radius is larger denote directions i about which we are more uncertain.

We provide (very hand-wavy) intuition for how we come up with these radii here. Consider the regression Gram matrix $\mathbf{V}_{t-1} := \sum_{s=1}^{t-1} \mathbf{A}_s \mathbf{A}_s^\top$. Directions that are “well-represented” by the actions taken thus far will be the ones for which $A_{s,i}$ will be large for many values of s (e.g. in the shortest path example, an edge is “well-represented” if it has been on many of the sample paths tried so far); thus, we will be quite certain about these actions. On the other hand, directions that have been barely explored will be the ones that we will be more uncertain about. To reflect this, we construct the *confidence ellipsoid*

$$\mathcal{C}_t := \{\theta : \|\theta - \hat{\theta}_t\|_{\mathbf{V}_{t-1}}^2 \leq \beta_t\} \quad (21.5)$$

for parameters β_t that we will not define here, but will only consider to be increasing with t . The intuition is that the regression Gram matrix \mathbf{V}_{t-1} rescales our levels of uncertainty along each direction in a way that’s proportional to how much we have seen that particular direction. Since \mathbf{V}_{t-1} grows with t in size, our overall uncertainty does reduce, but unevenly across different directions depending on how often they were “seen” by the chosen actions $\{\mathbf{A}_s\}_{s=1}^{t-1}$.

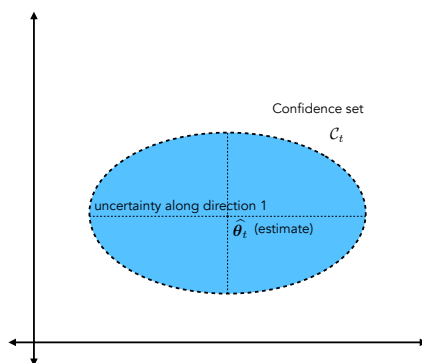


Figure 21.2: An illustration of the confidence set used in LinUCB for the case $d = 2$. This would be a situation in which \mathbf{V}_{t-1} is a diagonal matrix, and there is more uncertainty along the first direction than the second.

Exploration-vs-exploitation? Finally, it may not seem obvious yet how to map Step 3 to an exploration-exploitation tradeoff. We now briefly see how this is the case. It turns out (your HW will explore this in more detail) that we can equivalently write Step 3 of the LinUCB algorithm as

$$\mathbf{A}_t := \arg \max_{\mathbf{a} \in \mathcal{A}} \left[\langle \hat{\boldsymbol{\theta}}_t, \mathbf{a} \rangle + \sqrt{\beta_t} \cdot \|\mathbf{a}\|_{\mathbf{V}_{t-1}^{-1}} \right].$$

The first parameter $\langle \hat{\boldsymbol{\theta}}_t, \mathbf{a} \rangle$ denotes an *exploitation term*, and the second term $\|\mathbf{a}\|_{\mathbf{V}_{t-1}^{-1}}$ denotes an *exploration term* that incentivizes exploration along directions that have been relatively “unseen” (as \mathbf{V}_{t-1}^{-1} will be very large along those axes). We do not go through the details of the regret analysis of LinUCB in this class—the application of Hoeffding bounds turns out to be a lot harder as the action set is now continuous, and more sophisticated sequential statistics techniques are required. However, the program for proving regret bounds turns out to be very similar to UCB. See the end of the lecture note for recommended references to learn more about LinUCB and how it works.

21.3. Additional references

- Chapter 19 of Lattimore and Szepesvári (2020) is an excellent reference for learning more about the LinUCB algorithm for stochastic linear bandits. Chapter 30 also discusses the combinatorial bandit formulation that we presented here, but for a more complex “adversarial” setting that we will touch upon next lecture.
- One of the most popular applications of the combinatorial bandit problem involves selection problems in advertisement placement: suppose we have m ads that we want to display and d locations to choose out of them. Which are the best locations and how should we optimize for this efficiently? If m is comparable to d , there are exponential in d possibilities; but the rewards are highly correlated across them. Thus, the algorithms

discussed today can be leveraged to efficiently optimize ad placement in a manner that is *linear* rather than *exponential* in the number of locations d .

References

Tor Lattimore and Csaba Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020.