**ECE 8803: Online Decision Making in Machine Learning** **Fall 2021**

# Lecture 17: October 25

*Lecturer: Vidya Muthukumar*



Last lecture, we discussed the Thompson sampling algorithm and its guarantees on performance. We saw that in addition to the $\mathcal{O}\left(\sum_{a \neq a^*} \frac{\log T}{\Delta_a}\right)$ style guarantee on pseudo-regret (that UCB also satisfies), Thompson sampling can attain regret guarantees that scale proportional to the *optimal-action-entropy* of our prior belief. In other words, if we believe that the optimal arm is almost certainly going to be 1, we can drive our regret heavily down.

It turns out that all of 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 (and Thompson sampling with the Beta prior, like we introduced) 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.

## 17.1. Example 1: Combinatorial structure

Suppose that we have moved to a new city, and we are commuting from home to work every morning[1]. 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 17.1 demonstrates, the mean travel time along an edge $e$ is denoted by $\theta_e$, and a path $\boldsymbol{e} := (e_1, \ldots, e_n)$ 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 $\boldsymbol{e}^* := (e_1^*, \ldots, e_n^*)$.

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

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

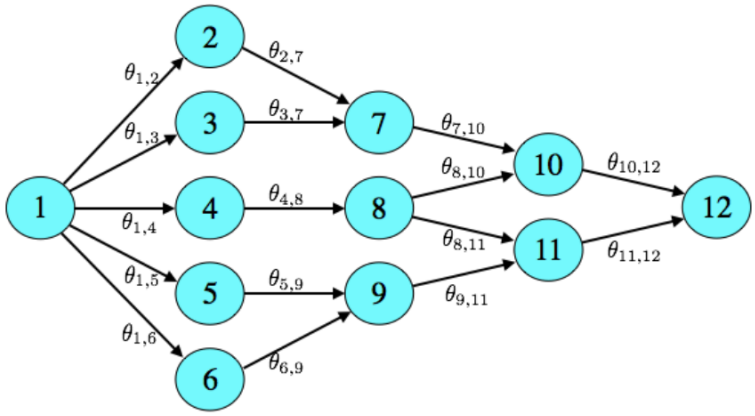



Figure 17.1: A depiction of the shortest-path problem with 12 vertices. Here, vertex 1 is the source vertex and 12 is the target (destination) vertex.

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.

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* $\boldsymbol{a} \in \mathcal{A}$ *is given by*

$$G_{t,a} = \langle \boldsymbol{a}, \boldsymbol{\theta^*} \rangle + W_t, \tag{17.1}$$

*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* $G_{t,a}$*, but it could additionally involve observing the individual components* $\theta^*_i$ *(along with some noise) corresponding to the indices* $i \in \{1, \ldots, d\}$ *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 17.1 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" $\boldsymbol{\theta}^*$ 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.

## 17.2. 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*

$$G_{t,a} = \langle \boldsymbol{a}, \boldsymbol{\theta}^* \rangle + W_t, \tag{17.2}$$

*where $W_t$ denotes some random noise and $\boldsymbol{\theta}^* \in \mathbb{R}^d$ 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 $\theta_e$ 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 $\boldsymbol{A}_t$. As before, pseudo-regret is defined with respect to the best arm $\boldsymbol{a}^* := \arg\max_{\boldsymbol{a}\in\mathcal{A}}\langle \boldsymbol{a}, \boldsymbol{\theta^*}\rangle$, i.e.

$$\overline{R}_T := \langle \boldsymbol{a}^*, \boldsymbol{\theta^*}\rangle - \mathbb{E}\left[\sum_{t=1}^{T} G_{t,\boldsymbol{A}_t}\right]. \tag{17.3}$$

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 $\overline{R}_T = \mathcal{O}(d\sqrt{T})$, 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.

### 17.2.1 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 $A_t$ 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* 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 $\widehat{\boldsymbol{\theta}}_t$.
- Step 2: Define a *confidence set* for this reward vector (that includes the sample mean $\widehat{\boldsymbol{\theta}}_t$), which we denote by $\mathcal{C}_t$. The confidence set is a subset of the action set $\mathcal{A}$ and is depicted in Figure 17.2. This generalizes the idea of a confidence interval.
- Step 3: Pick $\boldsymbol{A}_t = \arg\max_{\boldsymbol{a}\in\mathcal{A}} \max_{\boldsymbol{\theta}\in\mathcal{C}_t}\langle \boldsymbol{a}, \boldsymbol{\theta}\rangle$. This generalizes the optimism principle, as for a given action $\boldsymbol{a}$, 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 $\widehat{\boldsymbol{\theta}}_t$:** At round $t$, note that we have taken actions $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_{t-1}$ and observed rewards $G_{1,\boldsymbol{A}_1}, \ldots, G_{t-1,\boldsymbol{A}_{t-1}}$. Moreover, for a round $s$ we have $G_{s,\boldsymbol{A}_s} = \langle \boldsymbol{A}_s, \boldsymbol{\theta^*}\rangle + W_s$. Therefore, we can think of the actions $\{\boldsymbol{A}_s\}_{s=1}^{t-1}$ as *covariates*, and the observed rewards $\{G_{s,\boldsymbol{A}_s}\}_{s=1}^{t-1}$ as *responses* in a linear model with the unknown parameter

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

$$\widehat{\boldsymbol{\theta}}_t := (\boldsymbol{V}_{t-1} + \lambda \boldsymbol{I})^{-1} \sum_{s=1}^{t-1} \boldsymbol{A}_s G_{s,\boldsymbol{A}_s}, \tag{17.4}$$

where $\boldsymbol{V}_{t-1} := \sum_{s=1}^{t-1} \boldsymbol{A}_s \boldsymbol{A}_s^\top$ 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* $\boldsymbol{V}_{t-1} = \boldsymbol{X}_{t-1}^\top \boldsymbol{X}_{t-1}$*, where* $\boldsymbol{X}_{t-1} := \begin{bmatrix} \boldsymbol{A}_1^\top \\ \boldsymbol{A}_2^\top \\ \vdots \\ \boldsymbol{A}_{t-1}^\top \end{bmatrix}$*; 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 $\widehat{\boldsymbol{\theta}}_t$. 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 $1 \to 2 \to 7 \to 10 \to 12$ 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 $\langle \widehat{\boldsymbol{\theta}}_t, \boldsymbol{a} \rangle$ 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 $\boldsymbol{\theta^*}$ along each direction $i = 1, \ldots, d$. As pictured in Figure 17.1, 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 $\theta_i^*$, 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 $\boldsymbol{V}_{t-1} := \sum_{s=1}^{t-1} \boldsymbol{A}_s \boldsymbol{A}_s^\top$. 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*

$$\mathcal{C}_t := \{\boldsymbol{\theta} : \|\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}_t\|_{\boldsymbol{V}_{t-1}}^2 \leq \beta_t\} \tag{17.5}$$

for parameters $\beta_t$ that we will not define here, but will only consider to be increasing with $t$. The intuition is that the regression Gram matrix $\boldsymbol{V}_{t-1}$ rescales our levels of uncertainty along each direction in a way that's proportional to how much we have seen that particular direction. Since $\boldsymbol{V}_{t-1}$ 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 $\{\boldsymbol{A}_s\}_{s=1}^{t-1}$.

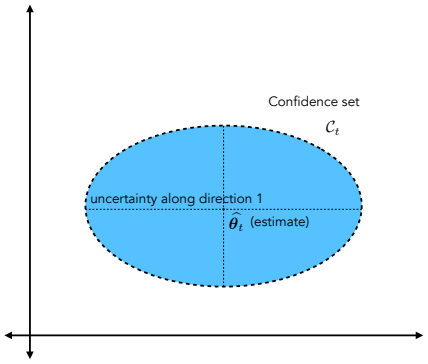



Figure 17.2: An illustration of the confidence set used in LinUCB for the case $d = 2$. This would be a situation in which $\boldsymbol{V}_{t-1}$ is a diagonal matrix, and there is more uncertainty along the first direction than the second.

**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

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

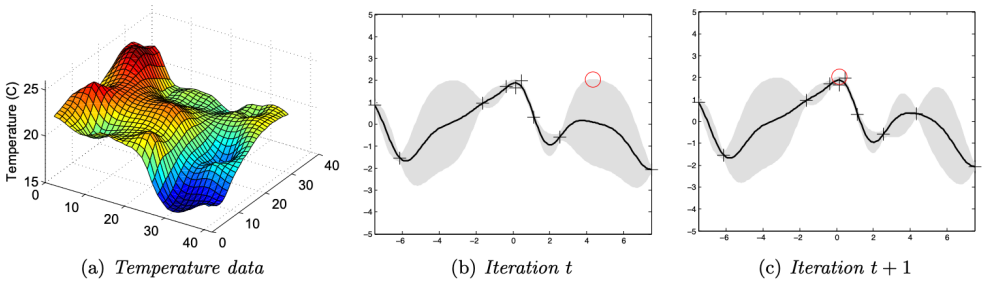
The first parameter $\langle \widehat{\boldsymbol{\theta}}_t, \boldsymbol{a} \rangle$ denotes an *exploitation term*, and the second term $\|\boldsymbol{a}\|_{\boldsymbol{V}_{t-1}^{-1}}$ denotes an *exploration term* that incentives exploration along directions that have been relatively "unseen" (as $\boldsymbol{V}_{t-1}^{-1}$ 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.

## 17.3. Applications of bandits: experiment design and Bayesian optimization

While the linear bandit formulation is a powerful generalization of MAB ideas to large-scale, structured bandit problems, it still does not characterize several real-world settings of interest. The most general version of a structured bandit problem would involve trying to find the *maximum* of an unknown reward function $g(\boldsymbol{a})$ over $\boldsymbol{a} \in \mathcal{A}$ by adaptively collected samples $\{\boldsymbol{A}_1, \ldots, \boldsymbol{A}_{s-1}\}$ and their corresponding (noisy) evaluations, given by

$$G_{s,\boldsymbol{A}_s} = g(\boldsymbol{A}_s) + W_s \tag{17.6}$$

Note that unlike in a classic optimization problem, we do not have access to the true function $g(\cdot)$, nor any information about its derivatives (e.g. the gradient); hence, we are solely using




(a) *Temperature data* (b) *Iteration t* (c) *Iteration t + 1*

Figure 17.3: The experiment design problem with Intel's sensor data. The left panel shows $g(\boldsymbol{a})$, and the right two panels depict consecutive runs of GP-UCB. The solid line denotes the estimate $\widehat{g}_t(\cdot)$ of the function at round $t$, and the shaded area denotes the regions of uncertainty. The middle panel shows a decision that prioritized exploration (as it chooses a location where the shaded area is wide), and the right-most panel shows a decision that prioritized exploitation (as the solid line is maximized there).

function evaluations to try and compute and arrive at the optimum $\boldsymbol{a}^* = \arg\max_{\boldsymbol{a}\in\mathcal{A}} g(\boldsymbol{a})$. In settings of *experiment design* where this problem often arises, we want to try and estimate $\boldsymbol{a}^*$ by conducting as few evaluations as possible. One of the earliest concrete examples of experiment design arises in geo-spatial sensing applications that are conducted real-time, we may want to place sensors to collect measurements, and locate and diagnose a "optimum point" as quickly as possible. For example, we may want to find the location of highest temperature in a large building (and then control for it) by sequentially activating sensors in a spatial network in the building and collecting temperature measurements. Here, $\mathcal{A}$ is the set of sensor configurations, $g(\boldsymbol{a})$ is the temperature measurement arising from sensor configuration $\boldsymbol{a}$, and $\boldsymbol{A}_t$ is the configuration that is chosen at round (measurement index) $t$. See the left-most subfigure in Figure 17.3 for a surface plot of $g(\boldsymbol{a})$ v.s. $\boldsymbol{a}$. Since each sensor activation draws battery power, we would want to take as few such measurements as possible. Similarly, we could deploy traffic sensors along a long interstate highway to identify the most congested portion of the highway, or the location where maximum speeding is occurring. To conduct effective interventions in traffic control, finding these locations as quickly as possible is paramount; hence, a bandit formulation is well-motivated. Moreover, just as in the linear bandit formulation, simply trying all configurations can take prohibitively long due to the curse of dimensionality that is involved.

Note that the linear bandit formulation is a special case of Equation (17.6) with $g(\boldsymbol{a}) = \langle \boldsymbol{\theta}^*, \boldsymbol{a} \rangle$ for a $d$-dimensional parameter. The experiment design problem that we describe above can be significantly more complicated in practice: after all, it is typically not the case that temperature or traffic measurements are linear in their locations. However, a different type of *local* structure may be more natural, and could be leveraged by a different type of bandit algorithm. For example, we would expect that nearby locations have correlated temperature measurements: if we take a measurement from location $\boldsymbol{a}_1$, we would hope to learn more about the evaluation at location $\boldsymbol{a}_2$ the closer $\boldsymbol{a}_2$ is to $\boldsymbol{a}_1$ in distance. Colloquially,

we expect our function $g(\cdot)$ to be *smooth*: $g(\boldsymbol{a}_2)$ cannot be too different from $g(\boldsymbol{a}_1)$. You can see this smoothness pattern manifest in Figure 17.3 for Intel's temperature sensor data (interestingly, little other helpful structure is present—the function does not appear to be concave, for example).

There are many ways to mathematically model this spatial structure. Here, we present a *Bayesian* approach, where (as in the MAB problem) we consider a **prior distribution** on the unknown function $g(\cdot)$. In particular, we consider the prior to be a *Gaussian process*, formally defined below.

**Definition 3** *A Gaussian process distribution on $g(\cdot)$ stipulates that for any location $\boldsymbol{a}$, the **prior distribution** of $g(\boldsymbol{a})$ is normal with mean $\mu(\boldsymbol{a})$ and variance $k(\boldsymbol{a};\boldsymbol{a})$. To model correlations, the covariance between two locations $\boldsymbol{a}_1, \boldsymbol{a}_2$ is given by $k(\boldsymbol{a}_1, \boldsymbol{a}_2) \neq 0$, where $k(\boldsymbol{a}_1, \boldsymbol{a}_2)$ is inversely proportional to the distance $\|\boldsymbol{a}_1 - \boldsymbol{a}_2\|$ and is commonly called a "kernel function".*

This "kernel function" is typically a modeling choice, just as the prior in Thompson sampling was a modeling choice. This Gaussian process model for the prior is very convenient, as it turns out that (just like in the Beta prior example we studied previously) the posterior distribution collected from samples is also a Gaussian process, and its parameters can be easily updated. Using this structure, versions of UCB and Thompson sampling can be generalized to this very general experiment design setting, and regret bounds can be proved. Such bounds will typically scale as $\mathcal{O}(\sqrt{T})$, and involve additional problem-complexity terms that are a function of the *kernel function* that is used to model the Gaussian process. For example, the regret bounds will turn out to be smaller, the larger the correlations are among the different locations.

The Gaussian process model allows us to mathematize this intuition in very general settings. Moreover, the generalizations of UCB and Thompson sampling can also be verified to satisfy an exploration-exploitation tradeoff. The right two sub-figures in Figure 17.3 depict two consecutive runs of UCB generalized to Gaussian processes (commonly called GP-UCB Srinivas et al. (2009)): one of the runs prioritizes exploration, and the other prioritizes exploitation. While we do not discuss the algorithm or proof techniques in detail here (they require an advanced machine learning/statistics background, particularly in kernel theory), these are excellent algorithms to investigate for either a theoretical or empirical course project: they have seen widespread application in engineering and experimental design settings. See the references below for further details.

## 17.4. 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.

- As we briefly discussed last lecture, Thompson sampling can also be adapted to work with structured linear bandit problems. See the tutorial paper on Thompson sampling Russo et al. (2018) and references therein for more details.

- 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 $d$.

- Several heuristics have existed for a long time for the experiment design/Bayesian optimization problem that we describe involving "expected improvement" or "most probable improvement". The first paper to leverage UCB-based ideas in this problem (and prove regret bounds) was Srinivas et al. (2009); ever since, GP-UCB and related bandit algorithms have seen widespread application in the settings of experiment design or Bayesian optimization.

- The applications of experiment design/Bayesian optimization are widespread and numerous and include sensor applications, control and even biology. In addition to the sensoring applications mentioned above, these ideas are used in reinforcement learning to flexibly model transition dynamics when overly simple models may not suffice, but we may not have sufficient amounts of data to train a model as complex as a deep neural network (Deisenroth et al., 2013). A particularly important application involves *safe* training in RL (e.g. Berkenkamp et al. (2017)): can we find a good policy as quickly as possible without trying heavily suboptimal policies, or find policies that flexibly adapt to changing environments? This is a critical property to be able to reliably deploy RL real-time, in high-stakes environments.

## References

Felix Berkenkamp, Matteo Turchetta, Angela P Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. *arXiv preprint arXiv:1705.08551*, 2017.

Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE transactions on pattern analysis and machine intelligence*, 37(2):408–423, 2013.

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

Daniel J Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen. A tutorial on thompson sampling. *Foundations and Trends® in Machine Learning*, 11(1): 1–96, 2018.

Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 2009.