Combinatorial Bandits for Sequential Learning in Colonel Blotto Games

Dong Quan Vu, Patrick Loiseau, Alonso Silva

Combinatorial Bandits for Sequential Learning in Colonel Blotto Games

Dong Quan Vu
AAAIRD Department
Nokia Bell Labs, Paris Saclay,
Nozay, France

Patrick Loiseau
Univ. Grenoble Alpes, Inria,
CNRS, Grenoble INP, LIG,
France & MPI-SWS, Germany

Alonso Silva
Safran Tech, Signal and
Information Technologies,
Magny-Les-Hameaux, France

***Abstract*— The Colonel Blotto game is a renowned resource allocation problem with a long-standing literature in game theory (almost 100 years). However, its scope of application is still restricted by the lack of studies on the incomplete-information situations where a learning model is needed. In this work, we propose and study a regret-minimization model where a learner repeatedly plays the Colonel Blotto game against several adversaries. At each stage, the learner distributes her budget of resources on a fixed number of battlefields to maximize the aggregate value of battlefields she wins; each battlefield being won if there is no adversary that has higher allocation. We focus on the bandit feedback setting. We first show that it can be modeled as a path planning problem. It is then possible to use the classical COMBAND algorithm to guarantee a sub-linear regret in terms of time horizon, but this entails two fundamental challenges: (i) the computation is inefficient due to the huge size of the action set, and (ii) the standard exploration distribution leads to a loose guarantee in practice. To address the first, we construct a modified algorithm that can be efficiently implemented by applying a dynamic programming technique called weight pushing; for the second, we propose methods optimizing the exploration distribution to improve the regret bound. Finally, we implement our proposed algorithm and perform numerical experiments that show the regret improvement in practice.**

I. INTRODUCTION

The *Colonel Blotto game* (henceforth, $\mathcal{CB}$ game) is a classical resource allocation problem where players simultaneously distribute a fixed number of troops (indivisible resources) on a certain number of battlefields to maximize the aggregate value of battlefields they win, each battlefield being won by the player who allocates the most resources to it. The $\mathcal{CB}$ game can be used to model a vast range of practical situations, e.g., allocating security forces in security ([1], [6]); allocating broadcasting time in advertisement ([15]); allocating shared spectrum in wireless networks ([11]) and allocating resources to persuade voters in politics ([12]).

The $\mathcal{CB}$ game was first introduced by [4] in 1921 and has been extensively studied after, especially in recent years. In particular, the Nash equilibrium of the relaxed continuous version (where resource allocations can be fractional) was studied by [8], [18], [20] under different assumptions. For the discrete version (with indivisible troops), [3] proposed polynomial (but still expensive) algorithms to compute the exact equilibrium whereas [23] proposed a much faster algorithm to compute an approximate equilibrium. All these works, however, only consider the full information one-shot game setting.[1] In most of the applications, the most natural setting is an incomplete information repeated game. For example, in advertising, one can consider an online marketing campaign which once per day reviews how the marketing campaign performs and based on this information, learns a better strategy. Resources and strategies of the adversaries are usually unknown or need to be estimated by the learner; especially in the dynamic setting. The question is then how to design a good sequential resource allocation policy.

The $\mathcal{CB}$ game has a specific combinatorial structure and the most natural way to model sequential learning in this game is to use the Combinatorial Bandit (henceforth, CBAND) framework, defined as follows: at each stage t within a time horizon T, the learner chooses a vector $\boldsymbol{p}^t$ in her action set $S \subset \{0,1\}^E$, for an $E \in \mathbb{N}$; then a loss vector $\boldsymbol{\ell}^t \in [0,1]^E$ is generated by the adversaries; the learner suffers a scalar loss $L(\boldsymbol{p}^t) = (\boldsymbol{\ell}^t)^\top \boldsymbol{p}^t$. The learner's objective is to minimize her *expected regret* R_T, i.e., the difference between her cumulative loss and that of her single best-action in hindsight, formally defined as follows:

$$R_T := \mathbb{E}\left[\sum_{t=1}^{T} L(\boldsymbol{p}_t) - \min_{\boldsymbol{p}\in S} \sum_{t=1}^{T} L(\boldsymbol{p})\right]. \tag{1}$$

Importantly, in CBAND, the feedback that the learner receives at the end of each stage is under the *bandit setting*: the learner's only observation when stage t ends is the scalar loss $L(\boldsymbol{p}^t)$. This is the most generic information-setting considered in the literature. This setting covers many applications of the $\mathcal{CB}$ game; e.g., in advertising, the total profit of selling a product can be easily observed while it is much harder to keep track of the partial profit of each ad (simultaneously promoting that product).

COMBAND algorithm, proposed by [5], is a classical algorithm for solving CBANDs. It provides a regret guarantee of the order $\mathcal{O}(\sqrt{TE\log|S|})$ that improves significantly than naively using other standard Multi-armed Bandit algorithms. In the $\mathcal{CB}$ game (modeled as a CBAND), E is polynomial in the number of battlefields and troops while $|S|$ is exponential in terms of these parameters. However, applying directly COMBAND, we face two important challenges.

a) Challenge 1: Computation Issue: The main problem with the COMBAND algorithm is that in general, it cannot be implemented efficiently (its running time is in $\mathcal{O}(|S|T)$). However, in some special cases, there are techniques that allow us to efficiently implement variants of this algorithm. The *path planning problem* (henceforth, PPP) is such an example: each action is equivalent to a path on a graph and

[1] A few works studied dynamic settings of the game, but only limited to two or three stages and they focus on asynchronous allocations ([9], [17]).

the loss of a chosen path equals to the aggregation of losses of edges on that path. In PPPs, [19] recently proposed an efficient variant of COMBAND running in $\mathcal{O}(E^2T)$ based on the weight-pushing technique (introduced by [21]). However, this algorithm has a redundancy in representation (involving 5 sub-algorithms) and is still non-trivial to be implemented. A direct application of COMBAND to the $\mathcal{CB}$ model is impractical; can we find a new representation of $\mathcal{CB}$ game to obtain a path planning model allowing an efficient implementation of COMBAND? If the answer is positive, we also desire a simpler representation of this efficient algorithm.

b) Challenge 2: Optimizing Exploration Distribution: COMBAND mixes an exploitation procedure (updated according to an unbiased loss estimator) with an *"exploration distribution"* on the action set. The regret bound given by COMBAND algorithm depends directly on the choice of an exploration distribution to be used. For PPPs (and also for $\mathcal{CB}$ games), the optimal exploration distribution remains unknown (this is an open question proposed by [5]).

Our Contributions: In this paper, we provide the first analysis of sequential learning in $\mathcal{CB}$ games. The action set of the $\mathcal{CB}$ game can be represented by a special graph; thus, we can model it as a PPP. Focusing on this model, our contribution is twofold: (i) Based on the weight pushing technique, we construct a simple algorithm, called EDGE(μ), that can be efficiently implemented and it guarantees a polynomial regret bound in terms of the $\mathcal{CB}$ game's parameters; (ii) We propose a fast method to compute an exploration distribution that can be used as the input of EDGE(μ) to improve the regret bound. Numerical experiments are conducted to illustrate this improvement, both in terms of the performance and the computation time.

Although in this work, we focus only on the $\mathcal{CB}$ game, note that our setting is more general. Our results can be extended to PPPs with general graphs that includes many other resources allocation games and multi-task online optimization. Note finally that [7] proposed another variant of COMBAND (mixing with ideas of the OSMD algorithm proposed by [2]), called the COMBEXP algorithm, that improves the complexity of COMBAND in several cases while maintaining the regret guarantees. However, PPP is not explicitly considered in [7] and it remains an open question whether any arbitrary instance of the PPP satisfies the condition such that COMBEXP can be efficiently implemented (i.e., the convex hull of the action set can be represented by a polynomial number of linear equations and linear inequalities). Therefore, COMBAND is still the state-of-the-art algorithm for our considering problem. Moreover, COMBEXP also uses the uniform exploration distribution that is sub-optimal in PPPs (see also [5]); thus, our second contribution in finding better exploration distributions is relevant.

Notation: Throughout the paper, we use bold symbols (e.g., $\boldsymbol{x}$) to denote (column) vectors with subscript indexes (e.g., x_i) to denote its elements. On the other hand, the superscript t refers to the stage and the notation $[k]$ denotes the set $\{1, 2, \ldots, k\}$, for any $k \in \mathbb{N} \setminus \{0\}$. In graphs, we use the notation $e \in \boldsymbol{p}$ to refer that the edge e belongs to the path $\boldsymbol{p}$. Finally, $\top$ denotes the transpose matrix/vector and $\mathbb{M}_{k\times k'}$ is the set of all real matrices with dimension $k \times k'$.

II. PRELIMINARIES

In this section, we review the standard COMBAND algorithm (proposed by [5]). We also highlight its drawbacks that need be improved. A pseudo-code of COMBAND, written in our notation, is given as Algorithm 1.

Algorithm 1: COMBAND(μ) for CBAND.

Input: $S \subset \{0,1\}^E$, $T \in \mathbb{N}$, $\gamma \in [0,1]$, $\eta > 0$, distribution μ on S.

1 $\forall \boldsymbol{p} \in S,\ w^1(\boldsymbol{p}) := 1.$
2 **for** $t = 1, 2, \ldots, T$ **do**
3 Adversaries choose the loss vector $\boldsymbol{\ell}^t$ (unobserved by the learner).
4 $\forall \boldsymbol{p} \in S,\ \nu^t(\boldsymbol{p}) := w_t(\boldsymbol{p}) / \sum_{\boldsymbol{q}\in S} [w_t(\boldsymbol{q})].$
5 Sample and play $\boldsymbol{p}^t$ according to $d^t(\boldsymbol{p}) = (1-\gamma)\nu^t(\boldsymbol{p}) + \gamma\mu(\boldsymbol{p}).$
6 Suffer and observe the loss $L(\boldsymbol{p}^t) = \left(\boldsymbol{\ell}^t\right)^\top \boldsymbol{p}^t \le 1.$
7 Compute $C^t := \mathbb{E}_{\boldsymbol{p}\sim d^t}[\boldsymbol{p}\boldsymbol{p}^\top] \in \mathbb{M}_{E\times E}$.
8 Compute the estimated loss $\hat{\boldsymbol{\ell}}^t := L(\boldsymbol{p}^t)\left(C_t^{-1}\boldsymbol{p}^t\right) = \left(\boldsymbol{\ell}^t(\boldsymbol{p}^t)^\top\right) C_t^{-1}\boldsymbol{p}^t.$
9 $\forall \boldsymbol{p} \in S,\ w^{t+1}(\boldsymbol{p}) := w^t(\boldsymbol{p}) e^{-\eta(\hat{\boldsymbol{\ell}}^t)^\top \boldsymbol{p}}.$

At each stage t, COMBAND keeps a weight $w^t(\boldsymbol{p})$ for each action $\boldsymbol{p}$ and it samples an action (line 5) from a distribution, called d^t, mixing between an "exploitation" distribution ν^t (normalization of the action weights) and an "exploration" distribution μ (unchanged over time). An unbiased estimator $\hat{\boldsymbol{\ell}}^t \in [0,1]^E$, based on the "co-occurrence" matrix $C^t := \mathbb{E}_{\boldsymbol{p}\sim d^t}[\boldsymbol{p}\boldsymbol{p}^\top] \in \mathbb{M}_{E\times E}$, is used to estimate the loss vector $\boldsymbol{\ell}^t$. Then, the action weights are updated by the exponential rule using these estimated losses (line 9).

In COMBAND, the exploration distribution μ is chosen a priori and it can be any arbitrary distribution on S such that S is spanned by the support of μ. Importantly, the performance guarantee of COMBAND depends directly on the choice of μ; to highlight this, we henceforth parameterize COMBAND with μ and use the notation COMBAND(μ). Consider the matrix $M(\mu) = \mathbb{E}_{\boldsymbol{p}\sim\mu}\left[\boldsymbol{p}\boldsymbol{p}^\top\right]$, we denote by $\lambda^*[M(\mu)]$ the *smallest nonzero eigenvalue* of $M(\mu)$ and let $n := \max\{\|\boldsymbol{p}\|_1, \boldsymbol{p} \in S\}$. An upper-bound of the expected regret given by this algorithm is stated as follows.

Theorem 2.1: In any CBAND *problem, the* COMBAND(μ) *algorithm with appropriate parameters yields an expected regret* $R_T \le 2\sqrt{[2n/(E \cdot \lambda^*[M(\mu)])]+1]\, TE \log(|S|)}$.

This theorem is extracted from Theorem 1 in [5] and is rewritten here under our notation. Trivially, the larger $\lambda^*[M(\mu)]$ is, the better the regret bound that COMBAND(μ) guarantees. The problem of optimizing μ and $\lambda^*[M(\mu)]$ in general CBANDs (and particularly for PPPs) remains an open question (see [5] for several positive examples). Regarding the computation complexity of COMBAND, given a time horizon T, it runs in $\mathcal{O}(|S| \cdot T)$. Since $|S|$ is exponential in

terms of E, it is inefficient to implement COMBAND. This is due to the *weights-updating step* (line 9), the *sampling step* (line 5) and the computation of the *co-occurrence matrix* (line 7). We will analyze these steps in Section IV and provide alternative procedures to improve them in the sequential learning model of Colonel Blotto games.

III. COMBINATORIAL BANDIT MODEL OF LEARNING IN COLONEL BLOTTO GAMES

We consider a sequential learning problem that involves a *learner*, A adversaries, n battlefields and a time horizon T ($n \geq 2$ and $T > 0$ are known by the learner, $A \geq 1$). Each battlefield $i \in [n]$ has a fixed value $b_i > 0$ (hidden from the learner) and we assume normalized values, that is $\sum_{i=1}^{n} b_i = 1$. At each stage $t \in [T]$, the learner faces a decision problem of distributing m troops ($m \geq 1$ is fixed) towards the battlefields while the adversaries simultaneously allocate theirs. The learner's allocations have to satisfy the budget constraint, that is she chooses a strategy $\boldsymbol{p}^t$ in the action set $S := \{\boldsymbol{p} \in \mathbb{N}^n : \sum_{i=1}^{n} p_i = m\}$.[2] For any $i \in [n]$, the element p_i of strategy $\boldsymbol{p}$ represents the quantity of troops she allocates to battlefield i. At the end of time t, the learner suffers a loss $L(\boldsymbol{p}^t)$ equal to the sum of values of battlefields that she loses, i.e., where there is at least one adversary having strictly higher local allocation than her. Without loss of generality, in case there are players who have tie allocations which are the highest in battlefield i, the value b_i is shared equally among them. When t ends, the learner observes the scalar number $L(\boldsymbol{p}^t)$ but she does not know which battlefield she lost or won nor the strategies that the adversaries used (the bandit feedback). Note that the incurred loss is bounded, i.e., $L(\boldsymbol{p}^t) \leq 1, \forall \boldsymbol{p}^t, \forall t$. The cumulative loss $\sum_{t=1}^{T} L(\boldsymbol{p}^t)$ is computed and the learner's objective is to minimize her expected regret (defined in (1)). Henceforth, we refer to this model as $\mathcal{CB}(m, n)$ problem.

It is important to note that the size of the learner's strategy set is $|S| = \binom{n+m-1}{n-1} = \mathcal{O}(2^{\min\{n-1,m\}})$, that is exponential in terms of m (or n). Our objective is to design an algorithm guaranteeing an expected regret that is sub-linear in T and polynomial in terms of m and n while its complexity is also polynomial in m, n, and T.

A. Layered Graph and Learning in $\mathcal{CB}$ Games as PPPs

In this section, we restate the formulation of $\mathcal{CB}(m, n)$ as a PPP that allows an efficient implementation of COMBAND. To do this, for each $\mathcal{CB}(m, n)$ game, we design a special directed acyclic graph, called $G_{m,n}$, such that there exists a one-to-one mapping between the action set S and the set of all paths of $G_{m,n}$; we call this the *layered graph*.[3] The illustration of an instance of such a graph is presented in Figure 1. The proof of existence of $G_{m,n}$ can be intuitively seen in Figure 1 and a formal definition is as follows (note that this definition is extracted from [24]).

Definition 3.1 (Layered Graph): The graph $G_{m,n}$ is a DAG that contains:

(i) $N := 2+(m+1)(n-1)$ nodes that are arranged into $n+1$ layers. Layer 0 has only one node $s:=(0,0)$, called the source node and Layer n contains one node $d:=(n,m)$, called the destination node. Each Layer $i \in [n-1]$ contains $m+1$ nodes whose labels are ordered from left to right by $(i,0),(i,1),\ldots,(i,m)$.

(ii) There are directed edges from the source node s to all nodes in Layer 1 and from all nodes in Layer $n-1$ to the destination node d. For any Layer $i \in [n-2]$, there is a directed edge from node (i,j_1) to node $(i+1,j_2)$ (of Layer $(i+1)$) if $0 \leq j_1 \leq j_2 \leq m$.

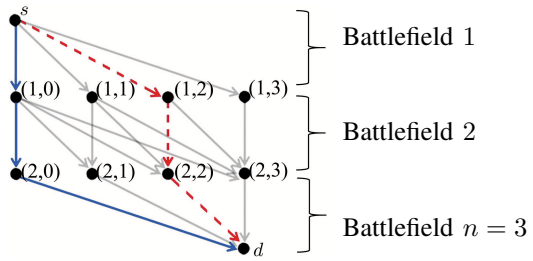


Fig. 1. The graph $G_{3,3}$ corresponding to the game with $m = n = 3$. Each path from s to d represents a strategy in S; e.g., the bold-blue path represents the strategy $(0,0,3)$ while the dash-red path represents the strategy $(2,0,1)$.

Let $\mathcal{N}$ denote the set containing all nodes of $G_{m,n}$ (including the source and destination nodes). $G_{m,n}$ has $E = (m+1)[4+(n-2)(m+2)]/2 = \mathcal{O}(nm^2)$ (directed) edges that are arranged into n layers. Each edge represents an allocation of a certain number of troops toward a certain battlefield: the edge from node (i,j_1) to node $(i+1,j_2)$ represents the allocation of player that puts (j_2-j_1) troops to battlefield $i+1$, for any $i \in \{0,\ldots,n-1\}$; for instance, in Figure 1, the edge from $(1,0)$ to $(2,3)$ represents allocating 3 troops to Battlefield 2. We denotes $\mathcal{E}$ the set containing all the edges. Hereinafter, referring to the layered graph, we simplify the notations and use the term "paths" to indicate the paths starting from s and ending at d if there is no other explicit explanation. We define the set $\mathcal{P} \subset \{0,1\}^E$ containing all such paths and $P := |\mathcal{P}| = |S| = \binom{n+m-1}{n-1} = \mathcal{O}(2^{\min\{n-1,m\}})$. Given a strategy $\boldsymbol{p} \in S \subset [0,m]^n$, we slightly abuse the notation and re-use $\boldsymbol{p} = (p_e)_{e\in\mathcal{E}}$ to denote the E-dimension 0-1-vector that represents the path equivalent to this strategy. Particularly, $\forall e \in \mathcal{E}$, $p_e = 1$ if and only if edge e belongs to path $\boldsymbol{p}$ and $p_e = 0$ otherwise.

Finally, for the sake of completeness, we write down formally the PPP that is equivalent to the $\mathcal{CB}(m, n)$ model. At each stage t, the allocations of the learner and the adversaries to the battlefields determine a scalar loss $\boldsymbol{\ell}_e^t \in [0,1]$ (following the rule of the $\mathcal{CB}$ game) embedded on each edge $e \in \mathcal{E}$ of the graph $G_{m,n}$. The learner has to choose a path $\boldsymbol{p}^t \in \mathcal{P} \subset \{0,1\}^E$ in $G_{m,n}$. The learner then suffers a loss $L(\boldsymbol{p}^t) = (\boldsymbol{\ell}^t)^\top \boldsymbol{p}^t = \sum_{e\in\boldsymbol{p}^t} \boldsymbol{\ell}_e^t$ which equals the sum

[2]The constraint $\sum_{i=1}^{n} p_i \leq m$ is sometimes considered in the literature. However, since unallocated troops do not contribute to the payoff, the learner always has incentives to use all her troops.

[3]This term is inspired by a graph proposed by [3] that looks similar to $G_{m,n}$; however, it is used for a completely different purpose and it also contains more edges and paths than $G_{m,n}$ (that are not useful in this work).

of losses from all edges belonging to the chosen path $\boldsymbol{p}^t$. At the end of stage t, the learner only observes the scalar loss $L(\boldsymbol{p}^t)$ of her chosen path but she does not know the loss of each edge. Henceforth, we focus our analysis on this model and we refer to it as PATHCB(m,n) to distinguish with $\mathcal{CB}(m,n)$.

IV. EFFICIENT ALGORITHM FOR PATH PLANNING PROBLEMS

In this section, we revisit a standard dynamic programming technique, called weight pushing. This technique is the basic for the efficient implementation of COMBAND algorithm in PPPs. The first idea of weight pushing technique could be tracked back to [10] and [21] although it was only applied to efficiently sample a path in PPPs according to updating-rules based on the weights of edges. Recently, [19] proposed an application of weight pushing to compute the co-occurrence matrix C^t in polynomial time in E; particularly for COMBAND in PPPs with the Zero-suppressed Binary Decision Diagrams. This computation requires 5 sub-algorithms that involves heavy notations and unneccessary complexity for our setting. We restate the technique with the notations and parameters of PATHCB(m,n) and propose a more computationally efficient version of COMBAND.

A. Paths and Edges' Weights

In PPPs, we call the action weights $w^t(\boldsymbol{p})$ involved in COMBAND as the *path weights* and recall that $w^{t+1}(\boldsymbol{p}) := w^t(\boldsymbol{p})e^{-\eta(\hat{\boldsymbol{\ell}}^t)^\top \boldsymbol{p}}$. At stage t, for each edge $e \in \mathcal{E}$, we define the *edge weight* w_e^t such that $w_e^1 := 1, \forall e$ (by convention) and $w_e^{t+1} := w_e^t \cdot e^{-\eta(\hat{\boldsymbol{\ell}}_e^t)}$. It is trivial to deduce that $w^t(\boldsymbol{p}) = \prod_{e\in\boldsymbol{p}} w_e^t$, $\forall \boldsymbol{p} \in \mathcal{P}, t \in [T]$, i.e., the weight of a path is the product of weights of all edges belong to it. The basic idea of weight pushing is to keep track of the paths weights (there is an exponential number of them) via the edges weights (only a polynomial number of them) by exploiting the structure of the graph.

Now, let us denote by $\mathcal{P}_{(u,v)}$ the set of all paths starting from node u and ending at node v. Then, for each pair of nodes $(u,v) \in \mathcal{N}\times\mathcal{N}$, at stage t, we define $H^t(u,v) := \sum_{\boldsymbol{p}\in\mathcal{P}_{(u,v)}} \prod_{e\in\boldsymbol{p}} w_e^t$. Intuitively, $H^t(u,v)$ is the sum of weights of all paths from node u to node v. Importantly, by conventionally setting $H^t(u,u) := 1, \forall u \in \mathcal{N}$ and $H^t(u,v) = 0$ if $\mathcal{P}_{(u,v)} = \emptyset$, we can compute all the quantities $H^t(u,v)$ in $\mathcal{O}(E)$ time via the following procedure (see also [10]): we first re-label the nodes set by $\mathcal{N}=\{s=u_0,u_1,\ldots,d=u_N\}$ such that if there exists an edge connecting u_i to u_j then $i<j$. Then, for any $v \in \mathcal{N}$, recursively for $u \in \{v-1, v-2, \ldots, s:=0\}$, we can compute $H^t(u,v)$.

B. Sampling by Edges' Weights

Inspired by Theorem 3 in [10], we can design an algorithm, denoted WP Algorithm (WP stands for weight pushing), that takes $w^t(e), e \in \mathcal{E}$ as inputs and outputs a path in $\mathcal{P}$. More importantly, the probability that a path $\boldsymbol{p}$ is an output of the WP Algorithm at stage t is exactly $\nu^t(\boldsymbol{p})$—the exploitation distribution defined in COMBAND. We rewrite this algorithm under our notation as Algorithm 2. In Algorithm 2, we denote by $e_{[u,v]}$ the edge connecting from node u to node v and by $\mathcal{C}(u) := \{u' > u : e_{[u,u']} \in \mathcal{E}\}$ the set of all direct children of u.

Algorithm 2: WP Algorithm: Sampling by edges' weights.

Input: $G_{m,n}$, $t \in [T]$, $w_e^t, \forall e \in \mathcal{E}$.

Initialize $\mathsf{Q} := \{0\}$, $u_0 = s$ and $k = 0$.

for $k \le n$ **do**

  Sample a node u_{k+1} from $\mathcal{C}(u_k)$ with probability $w^t_{e_{[u_k,u_{k+1}]}} H^t(u_{k+1},d)/H^t(u_k,d)$.

  Add u_{k+1} to the set Q.

Output: $\boldsymbol{p}^t \in \mathcal{P}$ going through all nodes in P.

C. Co-occurrence Matrix Computation

We now turn our focus to the matrix $C^t := \mathbb{E}_{\boldsymbol{p}\sim d^t}[\boldsymbol{p}\boldsymbol{p}^\top]$ needed to be computed at each stage t in the COMBAND(μ) algorithm. A direct computation of this matrix involves a sum of P terms, that leads to the inefficiency of COMBAND(μ). We first consider the following assumption on μ:

Assumption 1: There exists a set of edges weights $\tilde{\boldsymbol{w}}_e > 0$, $e \in \mathcal{E}$ such that for each path $\boldsymbol{p}^* \in \mathcal{P}$, we have $\mu(\boldsymbol{p}^*) = \prod_{e\in\boldsymbol{p}^*} \tilde{w}_e / \sum_{\boldsymbol{p}\in\mathcal{P}} (\prod_{e\in\boldsymbol{p}} \tilde{w}_e)$.

Intuitively, if μ satisfies Assumption 1, there exists a set of edges weights such that each path weight (according to μ) equals to the multiplication of the weights of the corresponding edges. Note that the uniform distribution on $\mathcal{P}$ (used by most of works in the literature) satisfies Assumption 1.

Now, we observe that each entry $C^t{}_{e_1,e_2}$ equals to the probability that a chosen path $\boldsymbol{p}^t \sim d^t$ contains both edges e_1 and e_2 (hence the name co-occurrence matrix). Formally, we have $C^t_{e_1,e_2} = \sum_{\boldsymbol{p}\in\mathcal{P}} d^t(\boldsymbol{p})\boldsymbol{p}_{e_1}\boldsymbol{p}_{e_2} = \sum_{\{\boldsymbol{p}:e_1,e_2\in\boldsymbol{p}\}} d^t(\boldsymbol{p})$. Now, we define $M(\nu^t) := \mathbb{E}_{\boldsymbol{p}\sim\nu^t(\boldsymbol{p})}[\boldsymbol{p}\boldsymbol{p}^\top]$ and $M(\mu) = \mathbb{E}_{\boldsymbol{p}\sim\mu(\boldsymbol{p})}[\boldsymbol{p}\boldsymbol{p}^\top]$—the co-occurrence matrices corresponding to distribution ν^t and μ, respectively. From the definition of $d^t(\boldsymbol{p})$ in COMBAND(μ), we can rewrite

$$C^t = (1-\gamma)M(\nu^t) + \gamma M(\mu). \tag{2}$$

Therefore, to efficiently compute C^t, we need to efficiently compute $M(\nu^t)$ and $M(\mu)$. We do this by designing an algorithm, called Algorithm 3, based on the quantities $H^t(u,v)$ computed in the previous section. Algorithm 3 runs in $\mathcal{O}(E^2)$ time. $M(\nu^t)$ can always be computed by Algorithm 3 with input $w_e^t, e \in \mathcal{E}$. On the other hand, if μ satisfies Assumption 1, $M(\mu)$ can also be computed by Algorithm 3.

Note that, we keep a generic notation in this algorithm: the input $\tilde{w}_e, e \in \mathcal{E}$ refers to any configuration of edges weights, not only those with the specific forms w_e^t under the exponential updating rule. The output $M(\mu_{\tilde{w}})$ is the co-occurrence matrix corresponding to the distribution $\mu_{\tilde{\boldsymbol{w}}}$ that draws a path $\boldsymbol{p}^*$ with probability

$$\mu_{\tilde{\boldsymbol{w}}}(\boldsymbol{p}^*) = \prod\nolimits_{e\in\boldsymbol{p}^*} \tilde{w}_e / \sum\nolimits_{\boldsymbol{p}\in\mathcal{P}} (\prod\nolimits_{e\in\boldsymbol{p}} \tilde{w}_e). \tag{3}$$

Algorithm 3: Co-occurrence matrix computation.

Input: $G_{m,n}$, $\tilde{w}_e, \forall e \in \mathcal{E}$.

Compute $H(u,v) := \sum_{\boldsymbol{p} \in \mathcal{P}_{(u,v)}} \prod_{e \in \boldsymbol{p}} \tilde{w}_e$, $\forall u, v \in \mathcal{N}$.

for $e_1 = e_{[u_1,v_1]} \in \mathcal{E}$ **do**

  $M(\mu_{\tilde{\boldsymbol{w}}})_{e_1,e_1} = \frac{H(s,u_1)\tilde{w}_{e_1}H(v_1,d)}{H(s,d)}$.

  for $e_2 = e_{[u_2,v_2]} \in \mathcal{E}, e_2 > e_1$ **do**

    $M(\mu_{\tilde{\boldsymbol{w}}})_{e_1,e_2} = \frac{H(s,u_1)\tilde{w}_{e_1}H(v_1,u_2)\tilde{w}_{e_2}H(v_2,d)}{H(s,d)}$.

for $e_1, e_2 \in \mathcal{E}, e_2 < e_1$ **do** $M(\mu_{\tilde{\boldsymbol{w}}})_{e_1,e_2} = M(\mu_{\tilde{\boldsymbol{w}}})_{e_2,e_1}$.

Output: The matrix $M(\mu_{\tilde{\boldsymbol{w}}})$.

In Algorithm 3, we also drop the superscript t in the notation of $H(u,v)$; these quantities can be efficiently computed (with inputs $\tilde{w}_e$) similar to $H^t(u,v)$ (with inputs w_e^t). The main intuition of Algorithm 3 is that if a path $\boldsymbol{p}$ contains an edge $e_1 = e_{[u_1,v_1]}$, then $\boldsymbol{p}$ also has to contain a path from node s to node u_1 and a path from node v_1 to node d. Similarly, if a path $\boldsymbol{p}$ simultaneously contains the edges $e_1 = e_{[u_1,v_1]}$ and $e_2 = e_{[u_2,v_2]}$, then $\boldsymbol{p}$ also contains a path from node s to node u_1, a path from node v_1 to node u_2 and a path from node v_2 to node d.

D. EDGE - *An Computationally Efficient Algorithm*

We now combine the techniques presented in the previous sections into a modified variant of COMBAND. This novel algorithm, denoted EDGE, works on edges instead of paths. We also parameterize EDGE(μ) with each corresponding exploration distribution μ. Its pseudo code is given in Algorithm 4. We conclude this section with the following proposition.

Algorithm 4: EDGE(μ) Algorithm for PATHCB.

Input: $m, n, T \in \mathbb{N}$,$\gamma \in [0,1], \eta > 0$, distribution μ.

1 $\forall e \in \mathcal{E}$, $w_e^1 := 1$.

2 **for** $t = 1, 2, \ldots, T$ **do**

3 Adversaries play (unobserved by the learner).

4 Sample β from Bernoulli distribution $\mathcal{B}(\gamma)$.

5 **if** $\beta = 0$ **then** sample a path $\boldsymbol{p}^t$ using the WP Algorithm with $\{w_e^t, e \in \mathcal{E}\}$.

6 **else** Sample a path $\boldsymbol{p}^t$ from distribution μ.

7 Suffer and observe the loss $L(\boldsymbol{p}^t) = \left(\boldsymbol{\ell}^t\right)^\top \boldsymbol{p}^t \leq 1$.

8 Compute $C^t := \mathbb{E}_{\boldsymbol{p} \sim d^t}[\boldsymbol{p}\boldsymbol{p}^\top]$ based on (2) and Algorithm 3.

9 Estimate loss $\hat{\boldsymbol{\ell}}^t = \left(\boldsymbol{\ell}^t(\boldsymbol{p}^t)^\top\right) C_t^{-1} \boldsymbol{p}^t$.

10 $\forall e \in \mathcal{E}$, $w_e^{t+1} := w_e^t \cdot e^{-\eta \hat{\boldsymbol{\ell}}_e^t}$.

Proposition 4.1: With the same choices of γ and η, the expected regret of EDGE(μ) *is equal to that of* COMBAND(μ) *in* PPPs*; thus,* EDGE *has the same regret bound as indicated in Theorem 2.1. Given a distribution μ on $\mathcal{P}$ that satisfies Assumption 1,* EDGE(μ) *runs in $\mathcal{O}(n^2 m^4 T)$; this is in contrast with* COMBAND(μ) *that runs in $\mathcal{O}(\exp(n)T)$.*

V. OPTIMIZING THE EXPLORATION DISTRIBUTION AND NUMERICAL EVALUATION

In this section, we investigate the exploration distribution μ that is used in both COMBAND(μ) and EDGE(μ). Recall the notation $\lambda^*[M]$ for the smallest non-zero eigenvalue of a matrix M. In Theorem 2.1, the regret bound is of order $\mathcal{O}(\lambda^*[M(\mu)]^{-1/2})$; therefore, to minimize this bound, we need to search for μ that maximizes $\lambda^*[M(\mu)]$. In several CBAND problems (see [5]), the *uniform distribution* on the action set, denoted μ_{uni}, was proven to yield an optimal choice to use in COMBAND. However, it is not the case for general PPPs and particularly for PATHCB(m,n): the eigenvalue $\lambda^*[M(\mu_{\text{uni}})]$ may be of order $\Omega(P^{-1})$ which yields a regret upper-bound that is exponentially large in terms of the number of edges (an example can be found in [5]). Nevertheless, in all previous works that apply the COMBAND algorithm to PPPs, e.g. [10] and [19], μ_{uni} is used. Moreover, since it requires that $\gamma \leq 1$, the bound given in Theorem 2.1 can only be obtained if $T \geq [n \log(P)] \,/\, \left[(\lambda^*[M(\mu)])^2 \left(\frac{E}{n} + \frac{2}{\lambda^*[M(\mu)]}\right)\right]$ (parameters tuned by [5]). If $\lambda^*[M(\mu)]$ is too small, COMBAND(μ) and EDGE(μ) can only work with extremely large time horizon T, which is impractical. For these reasons, the choice of exploration distribution to use in these algorithms is crucial.

Formally, let us label the paths in $\mathcal{P}$ by $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_P$, we consider an eigenvalue-optimization problem as follows (its search space is P-dimensional):

$$\text{maximize} \quad \lambda^* \left[\sum\nolimits_{i=1}^{P} x_i \cdot \left[\boldsymbol{p}_i \boldsymbol{p}_i^\top\right]\right] \tag{4}$$

$$\text{subject to} \quad \boldsymbol{x} \in [0,1]^P, \sum\nolimits_{i=1}^{P} x_i = 1. \tag{5}$$

It is suggested in [5] that the problem (4)-(5) can be solved by casting it into a semi-definite programming problem (SDP). An explicit formulation of this SDP can be found in Appendix VII-A. In principle, this SDP can be solved exactly to find a distribution μ that maximizes $\lambda^*[M(\mu)]$. However, in practice, this SDP formulation still cannot be solved efficiently due to the fact that the feasible set still has dimension P and that it contains a constraint relating to a summation of P terms. In our simulation, standard SDP solvers[4] take a long running time to solve this SDP problem even with small instances and they easily run into computationally memory issues with moderate instances.

A. Derivative-free Optimization and Change of Search Space

The challenge is to find a fast method that provides an exploration distribution μ to be used in EDGE(μ) that guarantees a sufficiently good regret-bound. Moreover, it is desired to be able to efficiently sample a path from μ (line 6 in Algorithm 4) and to efficiently compute the matrix $M(\mu)$ in order to compute C^t (line 8 in Algorithm 4). To do this, we reformulate the problem (4)-(5) to reduce the dimension of

[4]CVXOPT solver, available at *https://cvxopt.org/* and Mosek solver *https://www.mosek.com/*, both use primal-dual interior points methods.

the search space. We consider the following problem whose search space is E-dimensional:

$$\max_{\boldsymbol{w}\in[0,\infty)^E} \lambda^*(M(\mu_{\tilde{\boldsymbol{w}}})). \tag{6}$$

Here, we recall the notation $\mu_{\tilde{\boldsymbol{w}}}$—the distribution on the paths set (defined in (3) for each $\tilde{\boldsymbol{w}} \in [0,\infty)^E$) such that each path weight is the multiplication of the corresponding edges weights. Therefore, for each feasible solution of (6), say $\boldsymbol{w}^*$, we can construct a corresponding feasible solution $\mu_{\boldsymbol{w}^*}$ of (4)-(5); moreover, the objective function of (6) at $\boldsymbol{w}^*$ equals to that of (4)-(5) at $\mu_{\boldsymbol{w}^*}$. The construction of $\mu_{\boldsymbol{w}^*}$ is in $\mathcal{O}(P)$ time, but we do not need to explicitly do so in order to run EDGE algorithm with $\mu_{\boldsymbol{w}^*}$. Instead, since $\mu_{\boldsymbol{w}^*}$ is guaranteed to satisfy Assumption 1, we can use the WP Algorithm to sample efficiently a path from $\mu_{\boldsymbol{w}^*}$ and use Algorithm 3 to compute efficiently $M(\mu_{\boldsymbol{w}^*})$. Therefore, we can solve (6) to (implicitly) find an exploration distribution and use it efficiently in EDGE.

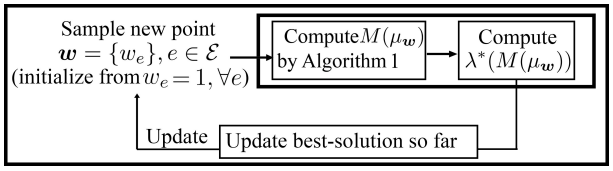


Fig. 2. Diagram illustrating the derivative-free optimization.

Although (6) reduces significantly the dimension of the search space comparing to (4)-(5), this formulation loses the structure that allows us to apply standard convex optimization algorithms.[5] Therefore, in this work, we use *derivative-free* algorithms to heuristically solve (6). Despite the fact that the solution found by this method may not be optimal, we can still guarantee that this solution is at least as good as the uniform distribution that is often used in the state-of-the-art algorithms (we initialize our algorithm with μ_{uni}). Moreover, although the search space in (6) may not cover the whole search space in (4)-(5), the solution found in (6) (might be corresponding to a sub-optimal for (4)-(5)) is guaranteed to be efficiently embedded with EDGE; on the other hand, even if we found an optimal solution of (4)-(5), it does not guarantee to be efficiently used in EDGE. A diagram explaining the intuition of our method to solve (6) can be found in Figure 2. We can use any derivative-free optimization solver that goes with specific strategies of sampling new points and justifying the current-best solution.

We denote μ_{free} the distribution corresponding to the solution of (6) found by our derivative-free method[6] and note that $\lambda^*(M(\mu_{\text{free}})) \geq \lambda^*(M(\mu_{\text{uni}}))$. Finally, as a corollary of Theorem 2.1 and Proposition 4.1, we have:

Proposition 5.1: In PATHCB(m,n)*, with appropriate parameters* γ *and* η*,* EDGE(μ_{free}) *guarantees* $R_T \leq \mathcal{O}\left(n\sqrt{\frac{2T}{\lambda^*[M(\mu_{free})]}}\right)$ *and runs in* $\mathcal{O}(n^2m^4T)$ *time.*

[5]The function giving the smallest non-zero eigenvalue of a matrix $M(\mu_{\boldsymbol{w}})$ from an input $\boldsymbol{w}$ is not known to be convex or concave.

[6]Take $\boldsymbol{w}_e = 1, \forall e \in \mathcal{E}$ (corresponding to μ_{uni}) as the initialization point.

B. Numerical Evaluation

We conduct several experiments to evaluate the performance of EDGE and measure the effect of optimizing the exploration distribution.[7] In these experiments, without loss of generality, a learner, having m troops, plays a repeated $\mathcal{CB}$ game on n battlefields with a single adversary who has m_A troops. We define a special adversary, called the *extreme-strong adversary*: an adversary having $m_A = (n-1)(m+1)+(m-1)$ troops, she "blocks" $n-1$ battlefields (each has a value equal to $\varepsilon/(n-1)$) by allocating $m+1$ troops to them and allocating $m-1$ troops to a certain battlefield i with value $b_i = 1-\varepsilon$ (unknown to the learner). In this case, the losses on all paths are always 1 except for the single path representing that the learner allocates all m troops to battlefield i; this path yields the loss ε. We choose this adversary to follow an example in [5] illustrating why μ_{uni} fails to guarantee a good regret bound in PPPs. The algorithms need to "explore" the low-loss path as soon as possible to reduce the regret.

We use the ZOOPT solver[8] (see [14]) embedded with sRACOS algorithm ([13]) as the derivative-free optimization solver to heuristically solve (6) (its output is μ_{free}). Our experiments run on an Intel Core i5-7300U CPU@ 2.60GHz and 8.00GB RAM. Each instance is run 5 times and the average results are reported.

In the first experiment, we compare the running time between COMBAND and EDGE and the results confirm that COMBAND takes exponential time while EDGE runs in polynomial time in terms of m and n; these results are reported in Figure 3 (the numbers of edges and paths in the corresponding $G_{m,n}$ are also reported for the sake of comparison).

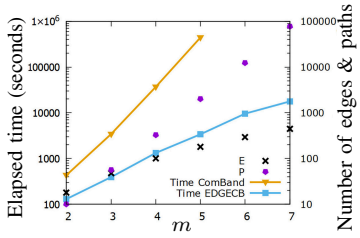
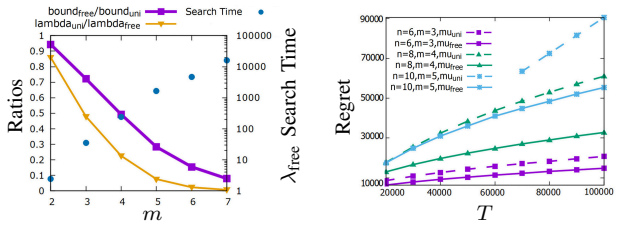


Fig. 3. COMBAND(μ_{free}) vs EDGE(μ_{free}); $n = 2m$, $T = 40000$ fixed.

Next, we compare the performances of EDGE when it uses μ_{uni} and μ_{free} as the exploration distribution. Figure 4(a) (y-axes is drawn with log-scale) illustrates the trade-off between the time spending to find μ_{free} and the improvement in the eigenvalues and the upper-bounds predicted by Theorem 2.1. Note that the smaller the ratios $\text{bound}_{\text{free}}/\text{bound}_{\text{uni}}$ and $\lambda_{\text{uni}}/\lambda_{\text{free}}$ are, the more improvement that EDGE(μ_{free}) provides comparing to EDGE(μ_{uni}). Finally, we compare the

[7]Our code is given at *https://github.com/dongquan11/BanditColonelBlotto*.

[8]Available at *https://zoopt.readthedocs.io/en/latest/*. We run it in $100E$ iterations; this stopping criterion is recommended by [13]; moreover, this criterion is enough to solve (6) optimally in our experiments with small instances ($m, n \leq 3$).



(a) $n=2m$, $T=40000$ fixed. (b) The actual regrets.

Fig. 4. Performances evaluation of EDGE(μ_{uni}) and EDGE(μ_{free}).

performance of EDGE(μ_{uni}) and EDGE(μ_{free}) by their actual regrets (see Figure 4(b)). Note that to efficiently compute the best hindsight loss (it is non-trivial), we apply a dynamic programming algorithm extracted from [23] that finds the best response against a set of allocations of the adversary. We observe that the actual regret of EDGE(μ_{free}) is better than EDGE(μ_{uni}); as m increases, the difference between these regrets also increases. For example, for instance $m=3$, $n=6$ and $T=10^5$, the ratio $(\text{Regret}_{\text{uni}}-\text{Regret}_{\text{free}})/\text{Regret}_{\text{uni}}$ equals 28% while this ratio of instance $m=5$, $n=10$, $T=10^5$ is 38%. Note that EDGE(μ_{free}) can run with larger instances (in m,n) but we choose not to report here since EDGE(μ_{uni}) is unavailable in these instances (it requires extremely large T). Besides the extreme-strong adversary, for this experiment, we also consider several other adversary's strategies (see Appendix VII-B for more details) and we notice that the results from these cases are similar to that of the extreme-strong adversary case.

VI. Conclusion

In this work, we present the EDGE algorithm for learning in the Colonel Blotto game that is modeled as a path planning problem. EDGE improves the regret guarantees compared to benchmark algorithm thanks to our proposed method finding an improved exploration distribution. Moreover, our algorithm is always efficiently implementable. This work not only extends the scope of application of the Colonel Blotto game in practice (even for large instances) but also can be applied to more general path planning problems.

References

[1] D. G. Arce, D. Kovenock, and B. Roberson, *Weakest-link attacker-defender games with multiple attack technologies*, Naval Research Logistics (NRL), 59 (2012), pp. 457–469.

[2] J.-Y. Audibert, S. Bubeck, and G. Lugosi, *Regret in online combinatorial optimization*, Mathematics of Operations Research, 39 (2014), pp. 31–45.

[3] S. Behnezhad, S. Dehghani, M. Derakhshan, M. Haji-Aghayi, and S. Seddighin, *Faster and simpler algorithm for optimal strategies of Blotto game.*, in AAAI, 2017, pp. 369–375.

[4] E. Borel, *La théorie du jeu et les équations intégrales à noyau symétrique*, Comptes rendus de l'Académie des Sciences, 173 (1921), p. 58.

[5] N. Cesa-Bianchi and G. Lugosi, *Combinatorial bandits*, Journal of Computer and System Sciences, 78 (2012), pp. 1404–1422.

[6] P. H. Chia and J. Chuang, *Colonel blotto in the phishing war*, in International Conference on Decision and Game Theory for Security, Springer, 2011, pp. 201–218.

[7] R. Combes, M. S. T. M. Shahi, A. Proutiere, et al., *Combinatorial bandits revisited*, in Advances in Neural Information Processing Systems, 2015, pp. 2116–2124.

[8] O. Gross and R. Wagner, *A continuous Colonel Blotto game*. U.S.Air Force Project RAND Research Memorandum, 1950.

[9] A. Gupta, G. Schwartz, C. Langbort, S. S. Sastry, and T. Bařar, *A three-stage colonel blotto game with applications to cyberphysical security*, in American Control Conference (ACC), 2014, IEEE, 2014, pp. 3820–3825.

[10] A. György, T. Linder, G. Lugosi, and G. Ottucsák, *The on-line shortest path problem under partial monitoring*, Journal of Machine Learning Research, 8 (2007), pp. 2369–2403.

[11] M. Hajimirsaadeghi and N. B. Mandayam, *A dynamic colonel blotto game model for spectrum sharing in wireless networks*, in 2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton), IEEE, 2017, pp. 287–294.

[12] R. Hortala-Vallve and A. Llorente-Saguer, *Pure strategy nash equilibria in non-zero sum Colonel Blotto games*, International Journal of Game Theory, 41 (2012), pp. 331–343.

[13] Y.-Q. Hu, H. Qian, and Y. Yu, *Sequential classification-based optimization for direct policy search.*, in AAAI, 2017, pp. 2029–2035.

[14] Y.-R. Liu, Y.-Q. Hu, H. Qian, Y. Yu, and C. Qian, *Zoopt/zoojl: Toolbox for derivative-free optimization*, arXiv preprint arXiv:1801.00329, (2017).

[15] A. M. Masucci and A. Silva, *Strategic resource allocation for competitive influence in social networks*, in Allerton, 2014, pp. 951–958.

[16] Y. Nesterov and A. Nemirovsky, *Interior-pointpolynomial methods in convexprogramming*, tech. rep., Philadelphia, PA,, 1994.

[17] R. Powell, *Sequential, nonzero-sum "blotto": Allocating defensive resources prior to attack*, Games and Economic Behavior, 67 (2009), pp. 611–615.

[18] B. Roberson, *The Colonel Blotto game*, Economic Theory, 29 (2006), pp. 2–24.

[19] S. Sakaue, M. Ishihata, and S.-i. Minato, *Efficient bandit combinatorial optimization algorithm with zero-suppressed binary decision diagrams*, in International Conference on Artificial Intelligence and Statistics, 2018, pp. 585–594.

[20] G. Schwartz, P. Loiseau, and S. S. Sastry, *The heterogeneous Colonel Blotto game*, in NetGCoop, 2014, pp. 232–238.

[21] E. Takimoto and M. K. Warmuth, *Path kernels and multiplicative updates*, Journal of Machine Learning Research, 4 (2003), pp. 773–818.

[22] L. Vandenberghe and S. Boyd, *Semidefinite programming*, SIAM review, 38 (1996), pp. 49–95.

[23] D. Q. Vu, P. Loiseau, and A. Silva, *Efficient computation of approximate equilibria in discrete Colonel Blotto games*, in IJCAI-ECAI, July 2018.

[24] D. Q. Vu, P. Loiseau, A. Silva, and L. Tran-Thanh, *Colonel blotto and hide-and-seek games as path planning problems with side observations*, arXiv preprint arXiv:1905.11151, (2019).

VII. Appendix

A. SDP Formulation of the Exploration-Distribution Optimization Problem

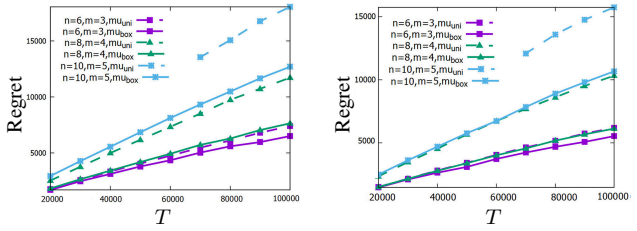
To formulate the problem (4)-(5) into a SDP, we first observe that for any distribution μ such that the paths set $\mathcal{P}$ is spanned by the support of μ, the matrix $M(\mu)$ always has a fixed number of zero eigenvalues (denoted K) and this number can be easily computed.[9] Therefore, the problem of maximizing $\lambda^*[M(\mu)]$ is equivalent to maximizing the sum of $K+1$ smallest eigenvalues of $M(\mu)$ which is formulated as:

$$\begin{aligned} \text{minimize} \quad & (K+1)s + \text{Tr}(Z) && (7)\\ \text{subject to} \quad & Z \succeq 0 && (8)\\ & Z + \sum\nolimits_{i=1}^{P} x_i \cdot \boldsymbol{p}_i \boldsymbol{p}_i^\top + sI_E \succeq 0. && (9) \end{aligned}$$

Here, $\boldsymbol{x} \in [0,1]^P$ and $r, s \in \mathbb{R}$, $Z \in \mathbb{M}_{E\times E}$ are the variables. I_E is the identity matrix and the notation $X \succeq 0$ indicates that the matrix X is positive semi-definite. This is trivially deduced from the Linear Matrix Inequalities representation of the sum of $K+1$ largest eigenvalues of the matrix (see e.g., [16], [22]).

B. Additional Numerical Experiments

Besides the extreme-strong adversary, we also consider two other adversary's strategies: the *uniform-adversary* (resp. the *battlefields-wise adversary*) who at each time t repeatedly draws a battlefield by uniform distribution (resp. draws battlefield i with probability $b_i / \sum_{j\in[n]} b_j$) then allocates one troop to that battlefield until he runs out of troops ($m_{\mathcal{A}} = m$). For this experiment, the battlefields' values b_i are generated uniformly from $[0, 8]$. For each instance with different parameters m, n and adversary's strategies, we run each algorithm EdgeCB(μ_{uni}) and EdgeCB(μ_{box}) 5 times and the average results of their actual regret are reported in Figure 5.



(a) Against uniform-adversary (b) Against battlefield-wise adversary

Fig. 5. The actual regrets of Edge(μ_{uni}) and Edge(μ_{free}).

[9] $Rank(M(\mu)) < E$ is the size of the largest linear independent subset of $\mathcal{P}$, which is fixed and only depends on the structure of the layered graph $G_{m,n}$. *Rank-nullity* theorem implies that K is also fixed. We can compute K by computing rank of any particular matrix, say $M(\mu_{\text{uni}})$.