ABSTRACT

Title of dissertation: CAMPAIGNING VIA LP'S
SOLVING BLOTTO AND BEYOND:

Saeed Seddighin
Doctor of Philosophy, 2019

Dissertation directed by: Professor MohammadTaghi Hajiaghayi
Department of Computer Science

The competition between the Republican and the Democrat nominees in the U.S presidential election is known as Colonel Blotto in game theory. In the classical Colonel Blotto game – introduced by Borel in 1921 – two colonels simultaneously distribute their troops across multiple battlefields. The outcome of each battlefield is determined by a winner-take-all rule, independently of other battlefields. In the original formulation, the goal of each colonel is to win as many battlefields as possible. The Colonel Blotto game and its extensions have been used in a wide range of applications from political campaigns (exemplified by the U.S presidential election) to marketing campaigns, from (innovative) technology competitions, to sports competitions. For almost a century, there have been persistent efforts for finding the optimal strategies of the Colonel Blotto game, however it was left unanswered whether the optimal strategies are polynomially tractable. In this thesis, we present several algorithms for solving Blotto games in polynomial time and will discuss their applications in practice.

CAMPAIGNING VIA LP'S SOLVING BLOTTO AND BEYOND

by

Saeed Seddighin

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2019

Advisory Committee:
Professor MohammadTaghi Hajiaghayi (chair)
Professor Lawrence M. Ausubel
Professor John Dickerson
Professor William Gasarch
Professor Christos Papadimitriou
Professor Shang-Hua Teng

Dedication

I would like to dedicate this thesis to my mother, brother, sister, and my late father for their unconditional love and support.

Acknowledgments

First and foremost, I would like to thank my adviser MohammadTaghi Hajiaghayi for all the time that we spent working together. Mohammad is indeed very smart, energetic, and hardworking. He has always made himself available for help and advice and there has never been an occasion when I knocked on his door and he did not give me time. I learned from Mohammad to be a fighter in research. Mohammad and I have worked on many challenging problems together[1] and I rarely ever saw him give up on finding a solution. Neither has he ever lost faith in any of his students during the ups and downs of a PhD life.

I would also like to thank Avrim Blum who has been a great source of motivation and inspiration for me during the time I had the pleasure of working with him at TTIC. Avrim is very knowledgeable, kind, and caring. I am very grateful for his continuous support.

My research has been greatly influenced by Christos Papadimitriou, Ronald Rivest, Clifford Stein, Shang-Hua Teng, Richard Karp, and Brendan Lucier. Thank you all for sharing your knowledge and expertise with me during our many fruitful discussions.

Many thanks to MohammadTaghi Hajiaghayi, Christos Papadimitriou, Shang-Hua Teng, Lawrence Ausubel, William Gasarch, and John Dickerson for devoting their time and serving on my thesis committee, and for their frequent advice and suggestions.

[1] I suspect that we may have set a new record high for the amount of adviser/student collaboration in a PhD.

The material of this thesis[2] is based on many discussions and fruitful collaborations with Amirmahdi Ahmadinejad, Soheil Behnezhad, Avrim Blum, Sina Dehghani, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Brendan Lucier, Mohammad Mahdian, Hamid Mahini, Christos Papadimitriou, Ronald Rivest, and Philip Stark. This thesis would not be possible without their help and directions.

I would like to thank my friends Melika Abolhassani, Saba Ahmadi, Roozbeh Basirian, Soheil Behnezhad, Ioana Bercea, Sina Dehghani, Mahsa Derakshan, Faeze Dorri, Soheil Ehsani, Hossein Esfandiari, Alireza Farhadi, Amin Ghiasi, Milad Gholami, Nima Hamidi, HamidReza Kazemitabar, Majeed Kazemitabar, MohammadReza Khani, Vahid Liaghat, Hamid Mahini, Morteza Monemizadeh, Mahyar Najibi, Kiana Roshanzamir, Hamed Saleh, Ali Shafahi, Mehrdad Tahvilian, Hadi Yami, and Bahar Zarrin[3] for making such great memories during the past five years.

Finally, I would like to thank my brother Masoud for being my biggest supporter from the age of 0. I would not have been where I am now had it not been for his selfless advice and generous support.

[2]These findings are also available in [1] (AAAI'16), [2] (AAAI'17), [3] (SODA'18), [4] (EC'18) and [5] (EC'19).

[3]Special thanks to Alireza Farhadi for sorting these names in the alphabetical order for me.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Game theory is the study of mathematical models of strategic interaction between rational decision-makers. The disciplines of game theory was founded in the early 20th century by famous mathematicians Ernst Zermelo and John von Neumann (1928). Game theory has had a major impact on the several branches of economics, social science, logic, and computer science. The field received considerable attention in computer science by the work of John Nash on the equilibria of games. John Nash introduced the concept of Nash equilibrium as a situation wherein no player benefits from changing his strategy unilaterally. His work on game theory is mostly known for his proof for the existence of Nash equilibria in all normal form games. Since then, computing such equilibria has become one of the most central problems in algorithmic game theory.

As mentioned earlier, it is known that every finite game admits a Nash equilibrium (that is, a profile of strategies from which no player can benefit from a unilateral deviation) [6]. However, it is not necessarily obvious how to find an equilibrium. Indeed, the conclusions to date have been largely negative: computing a Nash equilibrium of a normal-form game is known to be PPAD-complete [7–10], even for two-player games [11]. In fact, it is PPAD-complete to find an $\frac{1}{n^{O(1)}}$ approxima-

tion to a Nash equilibrium [12]. These results call into question the predictiveness of Nash equilibrium as a solution concept. This motivates the study of classes of games for which equilibria can be computed efficiently. It has been found that many natural and important classes of games have structure that can be exploited to admit computational results [13–16]. Perhaps the most well-known example is the class of zero-sum two-player games[1], where player 2's payoff is the negation of player 1's payoff. The normal-form representation of a zero-sum game is a matrix A, which specifies the game payoffs for player 1. This is a very natural class of games, as it models perfect competition between two parties. Given the payoff matrix for a zero-sum game as input, a Nash equilibrium can be computed in polynomial time, and hence time polynomial in the number of pure strategies available to each player [13]. Yet even for zero-sum games, this algorithmic result is often unsatisfactory. The issue is that for many games the most natural representation is more succinct than simply listing a payoff matrix, so that the number of strategies is actually exponential in the most natural input size. In this case the algorithm described above fails to guarantee efficient computation of equilibria, and alternative approaches are required.

1.1 The Colonel Blotto Game

A classical and important example illustrating these issues is the Colonel Blotto game, first introduced by Borel in 1921 [17–21]. In the Colonel Blotto game, two colonels each have a pool of troops and must fight against each other over a set

[1] Or, equivalently, constant-sum games.

of battlefields. The colonels simultaneously divide their troops between the battlefields. A colonel wins a battlefield if the number of his troops dominates the number of troops of his opponent. The final payoff of each colonel is the (weighted) number of battlefields won.

An equilibrium of the game is a pair of colonels' strategies, which is a (potentially randomized) distribution of troops across battlefields, such that no colonel has incentive to change his strategy. Although the Colonel Blotto game was initially proposed to study a war situation, it has found applications in the analysis of many different forms of competition: from sports, to advertisement, to politics [22–27], and has thus become one of the most well-known games in classic game theory.

Colonel Blotto is a zero-sum game. However, the number of strategies in the Colonel Blotto game is exponential in its natural representation. After all, there are $\binom{n+k-1}{k-1}$ ways to partition n troops among k battlefields. The classical methods for computing the equilibra of a zero-sum game therefore do not yield computationally efficient results. Moreover, significant effort has been made in the economics literature to understand the structure of equilibria of the Colonel Blotto game, i.e., by solving for equilibrium explicitly [27–37]. Despite this effort, progress remains sparse. Much of the existing work considers a continuous relaxation of the problem where troops are divisible, and for this relaxation a significant breakthrough came only quite recently in the seminal work of Roberson [34], 85 years after the introduction of the game. Roberson finds an equilibrium solution for the continuous version of the game, in the special case that all battlefields have the same weight. The more general weighted version of the problem remains open, as does the original

non-relaxed version with discrete strategies. Given the apparent difficulty of solving for equilibrium explicitly, it is natural to revisit the equilibrium computation problem for Colonel Blotto games. As part of this thesis, we show that Colonel Blotto admits a polynomial time solution in its most general form.

This work received a lot of attention in the U.S medial and elsewhere (see e.g. NSF [38], DARPA [39], Business Insider [40], Europapress [41] (in Spanish), Tivi [42] (in Finnish), Forskning [43] (in Norwegian), Mandiner [44] (in Hungarian), etc.).

"Colonel Blotto met its match in a team from @UMDScience supporting DARPA's GRAPHS program" (DARPA)

"And while variations of the game have been solved, until now no one has been able to find a way to arrive at a definitive solution for a two-party scenario. Now, a team from the University of Maryland, Stanford University and Microsoft Research says it has developed a generalized algorithm, which can now be applied to specific scenarios, such as the 2016 presidential election." (Business Insider)

"This solution allowed the team from UMD to develop a generalized algorithm, which can now be applied to specific situations, such as the 2016 presidential election." (Europapress, translated from Spanish)

1.2 A Faster Algorithm for Blotto

The ideas explained in Chapter 2 show how one can find the optimal strategies of Blotto in polynomial time even though the strategy space is exponentially large.

With this approach, the solution is found in three steps: 1) transferring the space of the game, 2) solving the game in the secondary space, and 3) translating the solution back to the primary space of the game. Steps 1 and 3 are necessary to reduce the size of the game as it is exponentially large in its current form. However, Step 2 deals with the computational barrier for solving Blotto. Roughly speaking, in this step we model the game with an LP with exponentially many constraints and we prove that using Ellipsoid method, one can optimize the LP in polynomial time.

This approach to solve Blotto involves sophisticated LP techniques such as the ellipsoid method. Although theoretically, ellipsoid method is a very powerful tool with deep consequences in complexity and optimization, it is "too inefficient to be used in practice" [45]. Interior point methods and simplex algorithm (even though it has exponential running-time in the worst case) are "far more efficient" [45]. Thus a practical algorithm for finding optimal strategies for the Blotto games remains an open problem. In fact, there have been huge studies in existence of efficient LP reformulations for different exponential-size LPs. For example Rothvoss [46] proved that the answer to the long-standing open problem, asking whether a graph's perfect matching polytope can be represented by an LP with a polynomial number of constraints, is negative. The seminal work of Applegate and Cohen [47] also provides polynomial-size LPs for finding an optimal oblivious routing. We provide a polynomial-size LP for finding the optimal strategies of the Colonel Blotto games. Although our original solution (Chapter 2) uses an LP with an exponential number of constraints, our new LP formulation (Chapter 3) has only $O(\mathsf{cnt}_{\mathsf{troops}}{}^2 k)$ constraints,

where $\mathsf{cnt}_{\mathsf{troops}} = a + b$ is the total number of troops and k denotes the number of battlefields. Consequently, we provide a novel, simpler, and significantly faster algorithm using the polynomial-size LP.

Furthermore, we show that our LP representation is asymptotically tight. The rough idea behind obtaining a polynomial-size LP is the following. Given a polytope P with exponentially many facets, we project P onto another polytope Q in a higher dimensional space which has a polynomial number of facets. Thus basically we are adding a few variables to the LP in order to reduce the number of constraints down to a polynomial. Q is called the *linear extension* of P. The minimum number of facets of any linear extension is called the *extension complexity*. We show that the extension complexity of the polytope of the optimal strategies of the Colonel Blotto game is $\Theta(\mathsf{cnt}_{\mathsf{troops}}{}^2 k)$. In other words, there exists no LP-formulation for the polytope of MaxMin strategies of the Colonel Blotto game with fewer than $\Theta(\mathsf{cnt}_{\mathsf{troops}}{}^2 k)$ constraints.

By implementing our LP, we observe the payoff of players in the continuous version considered by Roberson [34] very well predicts the outcome of the game in the auctionary and symmetric version of our model.

1.3 Maximizing the Likelihood of Winning an Election

The U.S. presidential election can be modeled as a Colonel Blotto game by corresponding each state to a battlefield and modeling the candidates' resources with the troops of the colonels. If the candidates were to maximize the *expected number*

of electoral votes, the optimal strategies could be characterized and computed via the algorithm presented in Chapter 2 (see also [1]). However, in the U.S. election, the goal of the parties is to maximize the likelihood of winning the race which is the probability that their candidate wins the majority of the electoral votes. To illustrate how different the two objectives could be, imagine that a strategy of a candidate secures an expected number of 280 electoral votes out of 538 votes in total. Now, if this solution guarantees 270 electoral votes with probability 0.5 and receives 290 electoral votes with probability 0.5, then the corresponding strategy always wins the race. Another (artificial) possibility is that this strategy receives 260 electoral votes with probability 9/10 and 460 votes with probability 1/10 and thus losing the race with probability 9/10 despite receiving more than half of the electoral votes in expectation.

Although expectation maximizer strategies of Blotto have received a lot of attention over the past few decades [17–20], prior to this, not much was known for the case where the goal is to maximize the likelihood of winning a certain amount of payoff. In this thesis, we also study this problem for both the discrete and continuous variants of Colonel Blotto. In particular, for the discrete variant of Colonel Blotto, we present a logarithmic approximation algorithm and improve this result for the continuous case to a constant approximation algorithm. We also present an exact algorithm for the guaranteed payoff setting (when the goal is to obtain a utility u with probability 1) in the continuous case. Moreover, we provide improved algorithms for the uniform case (when all the battlefields have the same

weight). Consider a two-player game between player A and player B.[2] We call a strategy of player A a (u, p)-maxmin strategy, if it guarantees a utility of at least u for her with probability at least p, regardless of the strategy that player B chooses. In other words, a strategy X is (u, p)-maxmin if for every (possibly mixed) strategy Y of the opponent we have

$$\Pr_{x \sim X, y \sim Y}[U^A(x, y) \geq u] \geq p,$$

where $U^A(x, y)$ denotes the payoff of player A if she plays strategy x and player B plays strategy y. Now for a given required payoff u and probability p, the problem is to find a (u, p)-maxmin strategy or report that there is no such strategy.

For many natural games, solving (u, p)-maxmin (for any given u and p) is computationally harder than the case where we focus on expected payoff (e.g., see Chapter 4 for a discussion about why it seems to be computationally harder to solve (u, p)-maxmin rather than maximizing expected payoff for Colonel Blotto). Therefore, it is reasonable to look for approximation algorithms. An approximate solution may relax the given probability, the given payoff, or both. We show in Chapter 4 that (u, p)-maxmin strategies can be approximated within provable guarantees in polynomial time.

[2] Our definition could easily be generalized to multi-player games.

1.4 Strategies with Small Profiles

Maximin strategies, or equivalently Nash equilibrium for constant-sum games such as Colonel Blotto, are often criticized for that they may be too complicated (see e.g., [48–50]). That is, even if we are able to find such a solution in polynomial time, we may not be able to deploy it since the equilibria can have a large support[3]. In the case of the Blotto game, the potential size of the support is enormous, while every possible pure strategy in the support requires a prior (often costly) setup. Therefore it is tempting to find a near-optimal strategy that uses very few pure strategies, and is near optimum against the opponent's best response — the last contribution of this thesis.

However, limiting the support size often renders the game intractable. For instance, the decision problem of existence of a Nash equilibrium when the support size is bounded by a given number is NP-hard even in two-player zero-sum games [51], while this problem is unlikely to be fixed parameter tractable when the problem is parameterized by the support size [52]. These results imply that in order to find optimal strategies with bounded support, we need to use structural properties of the game at hand, even when the players have polynomially many strategies. It becomes even more challenging for succinct games such as Colonel Blotto wherein the strategy space itself is exponentially large.

Note that even the unbounded case is challenging to solve as discussed earlier. In spite of that, for maximizing the expected utility, one can obtain optimal

[3]The *support* of a mixed strategy is the set of all pure strategies to which a non-zero probability is associated.

strategies in polynomial time by algorithms of Chapters 2 and 4.

All of the results explained in Chapter 4 rely crucially on the fact that the support size is unbounded. The challenges in obtaining bounded support strategies turn out to be entirely different. On one hand, for the choice of each pure strategy in the support we still have exponentially[4] many possibilities. On the other hand, we show that bounding the support size makes the solution space non-convex. The latter prevents us from using convex programming techniques in finding optimal strategies — which are essentially the main tools that are used in the literature for solving succinct games in polynomial time [1, 3, 53, 54]. However, we show through a set of structural results that the solution space can, interestingly, be partitioned into polynomially many disjoint convex polytopes, allowing us to consider each of these sub-polytopes independently. This leads to polynomial time approximation schemes (PTASs) for both the expectation case and the case of (u, p)-maxmin strategies.

We also provide in Chapter 5 the first complexity result for finding the maximin of Blotto-like games: we show that computing the maximin of a generalization of the Colonel Blotto game that we call General Colonel Blotto — roughly, the utility is a general function of the two allocations — is exponential time-complete.

1.5 Future Work and Open Problems

Indeed, improving the approximation factors of the algorithms of Chapters 4 and 5 are interesting directions for future work. Moreover, a particularly realistic

[4] Or even an unbounded number of strategies for the continuous variant of Colonel Blotto where the troops are capable of fractional partitioning.

variant of the Blotto game which we still do not know how to solve is when each troop contributes to each battlefield differently. Although our representation in this case remains convex and the utility functions are bilinear, finding a best-response of this game is NP-hard and therefore our framework does not provide a polynomial time solution.

Perhaps a broader and more relevant question that can be answered as future work is "to which games the presented framework apply?". Many classic zero-sum games have exponentially many strategies for players. Examples are dueling games, security games, parity games, stochastic games, poker heads up, etc. Apart from dueling and security games, our knowledge of the computational complexity of the rest of the games is limited. In particular, parity and stochastic games are proven to be in the intersection of NP and CoNP and therefore investigating their computational complexity is a very old and fundamental open question not only in game theory but in TCS in general.

In Chapter 2 we study the computational complexity of Blotto. This is essentially the main result of this thesis. In the following chapters, we study different variants: Chapter 3 discusses a faster algorithm for Blotto and Chapters 4 and 5 consider probabilistic versions of Blotto.

Chapter 2: Polynomial Time Solution for Colonel Blotto

We present a general method for computing Nash equilibria of a broad class of zero-sum games. Our approach is to reduce the problem of computing equilibria of a given game to the problem of optimizing linear functions over the space of strategies in a payoff-equivalent bilinear game.

Before presenting our general reduction, we will first illustrate our techniques by considering the Colonel Blotto game as a specific example. In Section 2 we describe our approach in detail for the Colonel Blotto game, explaining the process by which equilibria can be computed. Then in Section 2 we will present the general reduction. Further applications of this technique are provided in Section 2.1 (for dueling games) and Section 2.3 (for the General Lotto game).

Here, we propose a polynomial-time algorithm for finding an equilibrium of discrete Colonel Blotto in its general form. We allow the game to be *asymmetric* across both the battlefields and the players. A game is *asymmetric across the battlefields* when different battlefields have different contributions to the outcome of the game, and a game is *asymmetric across the players* when two players have different number of troops.

In the Colonel Blotto game, two players A and B simultaneously distribute

a and b troops, respectively, over k battlefields. A pure strategy of player A is a k-partition $x = \langle x_1, x_2, \ldots, x_k \rangle$ where $\sum_{i=1}^{k} x_i = a$, and a pure strategy of player B is a k-partition $y = \langle y_1, y_2, \ldots, y_k \rangle$ where $\sum_{i=1}^{k} y_i = b$. Let $u_i^A(x_i, y_i)$ and $u_i^B(x_i, y_i)$ be the payoff of player A and player B from the i-th battlefield, respectively. Note that the payoff functions of the i-th battlefield, u_i^A and u_i^B, have $(a+1) \times (b+1)$ entries. This means the size of input is $\Theta(kab)$. Since Colonel Blotto is a zero-sum game, we have $u_i^A(x_i, y_i) = -u_i^B(x_i, y_i)$[1]. Note that we do not need to put any constraint on the payoff functions, and our result works for all payoff functions. We also represent the total payoff of player A and player B by $h_{\mathcal{B}}^A(x, y) = \sum_i u_i^A(x_i, y_i)$ and $h_{\mathcal{B}}^B(x, y) = \sum_i u_i^B(x_i, y_i)$, respectively. A mixed strategy of each player would be a probability distribution over his pure strategies.

Theorem 1. *One can compute an equilibrium of any Colonel Blotto game in polynomial time.*

Proof. Let $\mathcal{X}$ and $\mathcal{Y}$ be the set of all pure strategies of players A and B respectively, i.e., each member of $\mathcal{X}$ is a k-partition of a troops and each member of $\mathcal{Y}$ is a k-partition of b troops. We represent a mixed strategy of player A with function $p : \mathcal{X} \to [0, 1]$ such that $\sum_{x \in \mathcal{X}} p(x) = 1$. Similarly, let function $q : \mathcal{Y} \to [0, 1]$ be a mixed strategy of player B. We may also use $\mathbf{x}$ and $\mathbf{y}$, instead of p and q, for referring to a mixed strategy of player A and B respectively. Since Colonel Blotto is a zero-sum game, we leverage the MinMax theorem for finding an NE of the game. This theorem says that pair (p^*, q^*) is an NE of the Colonel Blotto game if

[1]Note that in the Colonel Blotto game if $u_i^A(x_i, y_i)$ is not necessarily equal to $-u_i^B(x_i, y_i)$ then a special case of this game with two battlefields can model an arbitrary 2-person normal-form game and thus finding a Nash Equilibrium would be PPAD-complete.

and only if strategies p^* and q^* maximize the guaranteed payoff of players A and B respectively [55]. Now, we are going to find strategy p^* of player A which maximizes his guaranteed payoff. The same technique can be used for finding q^*. It is known that for each mixed strategy p, at least one of the best-response strategies to p is a pure strategy. Therefore, a solution to the following program characterizes strategy p^*.

$$\begin{aligned} \max \quad & U \\ s.t. \quad & \textstyle\sum_{x\in\mathcal{X}} p_x = 1, \\ & \textstyle\sum_{x\in\mathcal{X}} p_x h^A_{\mathcal{B}}(x,y) \geq U, \quad \forall y \in \mathcal{Y}, \end{aligned} \tag{2.1}$$

Unfortunately, LP 2.1 has $|\mathcal{X}|$ variables and $|\mathcal{Y}|+1$ constraints where $|\mathcal{X}|$ and $|\mathcal{Y}|+1$ are exponential. We therefore cannot solve LP 2.1 directly.

Step 1: Transferring to a new space. We address this issue by transforming the solution space to a new space in which an LP equivalent to LP 2.1 becomes tractable (See, e.g., [56], for similar technique). This new space will project mixed strategies onto the marginal probabilities for each (battlefield, troop count) pair. For each pure strategy $x \in \mathcal{X}$ of player A, we map it to a point in $\{0,1\}^{n(A)}$ where $n(A) = k\times(a+1)$. For convenience, we may abuse the notation, and index each point $\hat{\mathbf{x}} \in \{0,1\}^{n(A)}$ by two indices i and j such that $\hat{x}_{i,j}$ represents $\hat{x}_{(i-1)(a+1)+j+1}$. Now we map a pure strategy x to $\mathcal{G}_A(x) = \hat{\mathbf{x}} \in \{0,1\}^{n(A)}$ such that $\hat{x}_{i,j} = 1$ if and only if $x_i = j$. In other words, if player A puts j troops in the i-th battlefield then $\hat{x}_{i,j} = 1$. Let $I_A = \{\hat{\mathbf{x}} \in \{0,1\}^{n(A)} | \exists x \in \mathcal{X}, \mathcal{G}_A(x) = \hat{\mathbf{x}}\}$ be the set of points in $\{0,1\}^{n(A)}$ which represent

pure strategies of player A. Let $\mathcal{M}(\mathcal{X})$ and $\mathcal{M}(\mathcal{Y})$ be the set of mixed strategies of players A and B respectively. Similarly, we map mixed strategy $\mathbf{x}$ to point $\mathcal{G}_A(\mathbf{x}) = \hat{\mathbf{x}} \in [0,1]^{n(A)}$ such that $\hat{x}_{i,j}$ represents the probability that mixed strategy $\mathbf{x}$ puts j troops in the i-th battlefield. Note that mapping $\mathcal{G}_A$ is not necessarily one-to-one nor onto, i.e., each point in $[0,1]^{n(A)}$ may be mapped to zero, one, or more than one strategies. Let $S_A = \{\hat{\mathbf{x}} \in [0,1]^{n(A)} | \exists \mathbf{x} \in \mathcal{M}(\mathcal{X}), \mathcal{G}_A(\mathbf{x}) = \hat{\mathbf{x}}\}$ be the set of points in $[0,1]^{n(A)}$ which represent at least one mixed strategy of player A. Similarly, we use function $\mathcal{G}_B$ to map each strategy of player B to a point in $[0,1]^{n(B)}$ where $n(B) = k \times (b+1)$, and define $I_B = \{\hat{\mathbf{y}} \in \{0,1\}^{n(B)} | \exists y \in \mathcal{Y}, \mathcal{G}_B(y) = \hat{\mathbf{y}}\}$ and $S_B = \{\hat{\mathbf{y}} \in [0,1]^{n(B)} | \exists \mathbf{y} \in \mathcal{M}(\mathcal{Y}), \mathcal{G}_B(\mathbf{y}) = \hat{\mathbf{y}}\}$.

Lemma 2. *Set S_A forms a convex polyhedron with an exponential number of vertices and facets.*

Now, we are ready to rewrite linear program 2.1 in the new space as follows.

$$\begin{aligned} \max \quad & U & & (2.2) \\ s.t. \quad & \hat{\mathbf{x}} \in S_A & & \text{(Membership constraint)} \\ & h^A_{\mathcal{B}}(\hat{\mathbf{x}}, \hat{\mathbf{y}}) \geq U, \quad \forall \hat{\mathbf{y}} \in I_B & & \text{(Payoff constraints)} \end{aligned}$$

where

$$h^A_{\mathcal{B}}(\hat{\mathbf{x}}, \hat{\mathbf{y}}) = \sum_{i=1}^{k} \sum_{t_a=0}^{a} \sum_{t_b=0}^{b} \hat{x}_{i,t_a} \hat{y}_{i,t_b} u_i^A(t_a, t_b)$$

is the expected payoff of player A.

Step 2: Solving LP 2.2. The modified LP above, LP 2.2, has exponentially many

constraints, but only polynomially many variables. One can therefore apply the Ellipsoid method to solve the LP, given a separation oracle that runs in polynomial time [57,58]. By the equivalence of separation and optimization [59], one can implement such a separation oracle given the ability to optimize linear functions over the polytopes S_A (for the membership constraints) and S_B (for the payoff constraints).

Stated more explicitly, given a sequence $c_0, c_1, \ldots, c_{k(m+1)}$, where k is the number of battlefields and m is the number of troops for a player, the required oracle must find a pure strategy $x = (x_1, x_2, \ldots, x_k) \in \mathcal{X}$ such that $\sum_{i=1}^{k} x_i = m$, and $\hat{\mathbf{x}} = \mathcal{G}(x)$ minimizes the following equation:

$$c_0 + \sum_{i=1}^{k(m+1)} c_i \hat{x}_i, \tag{2.3}$$

and similarly for polytope $\mathcal{Y}$. The following lemma shows that one can indeed find a minimizer of Equation (2.3) in polynomial time.

Lemma 3. *Given two integers m and k and a sequence $c_0, c_1, \ldots, c_{k(m+1)}$, one can find (in polynomial time) an optimal pure strategy $x = (x_1, x_2, \ldots, x_k)$ where $\sum_{i=1}^{k} x_i = m$, $\hat{\mathbf{x}} = \mathcal{G}(x)$ and $\hat{\mathbf{x}}$ minimizes $c_0 + \sum_{i=1}^{k(m+1)} c_i \hat{x}_i$.*

Proof. We employ a dynamic programming approach. Define $d[i,t]$ to be the minimum possible value of $c_0 + \sum_{i'=1}^{i(t+1)} c_{i'} \hat{x}_{i'}$ where $\sum_{i'=1}^{i} x_{i'} = t$. Hence, $d[k,m]$ denotes the minimum possible value of $c_0 + \sum_{i=1}^{k(m+1)} c_i \hat{x}_i$. We have that $d[0,j]$ is equal to c_0 for all j. For an arbitrary $i > 0$ and t, the optimal strategy x puts $0 \leq t' \leq t$ units in the i-th battlefield and the applied cost in the equation 2.3 is equal to

$c_{(i-1)(m+1)+t'+1}$. Thus, we can express $d[i,t]$ as

$$d[i,t] = \min_{0 \le t' \le t} \{d[i-1, t-t'] + c_{(i-1)(m+1)+t'+1}\}.$$

Solving this dynamic program, we can find the allocation of troops that minimizes $\sum \alpha_i \hat{\mathbf{x}}_i$ in polynomial time, as required. □

Step 3: Transferring to the original space. At last we should transfer the solution of LP 2.2 to the original space. In particular, we are given a point $\hat{\mathbf{x}} \in S_A$ and our goal is to find a strategy $\mathbf{x} \in \mathcal{M}(\mathcal{X})$ such that $\mathcal{G}_A(\mathbf{x}) = \hat{\mathbf{x}}$. To achieve this, we invoke a classic result of [59] which states that an interior point of an n-dimensional polytope P can be decomposed as a convex combination of at most $n+1$ extreme points of P, in polynomial time, given an oracle that optimizes linear functions over P. Note that this is precisely the oracle required for Step 2, above. Applying this result to the solution of LP 2.2 in polytope S_A, we obtain a convex decomposition of $\hat{\mathbf{x}}$ into extreme points of S_A, say $\hat{\mathbf{x}} = \sum_i \alpha_i \hat{\mathbf{x}}_i$. Since each $\hat{\mathbf{x}}_i$ corresponds to a pure strategy in $\mathcal{X}$, it is trivial to find point $\mathbf{x}_i$ with $\mathcal{G}_A(\mathbf{x}_i) = \hat{\mathbf{x}}_i$, since the marginals of each $\hat{\mathbf{x}}_i$ lie in $\{0,1\}$. We then have that $\mathbf{x} = \sum_i \alpha_i \mathbf{x}_i$ is the required mixed strategy profile.

Combining these three steps, we find a Nash Equilibrium of the Colonel Blotto game in polynomial time, completing the proof of Theorem 1. See Section 2.2 for more details. □

In our method for finding a Nash Equilibrium of the Colonel Blotto game, the main steps were to express the game as a bilinear game of polynomial dimension,

solve for an equilibrium of the bilinear game, then express that point as an equilibrium of the original game. To implement the final two steps, it sufficed to show how to optimize linear functions over the polytope of strategies in the bilinear game. This suggests a general reduction, where the equilibrium computation problem is reduced to finding the appropriate bilinear game and implementing the required optimization algorithm. In other words, the method for computing Nash equilibria applies to a zero-sum game when:

1. One can transfer each strategy x of player A to $\mathcal{G}_A(x) = \hat{\mathbf{x}} \in R^{n(A)}$, and each strategy y of player B to $\mathcal{G}_B(y) = \hat{\mathbf{y}} \in R^{n(B)}$ such that the payoff of the game for strategies $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ can be represented in a bilinear form based on $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$, i.e., the payoff is $\hat{\mathbf{x}}^t M \hat{\mathbf{y}}$ where M is a $n(A) \times n(B)$ matrix.
2. For any given vector α and real number α_0 we can find, in polynomial time, whether there is a pure strategy $\hat{\mathbf{x}}$ in the transferred space such that $\alpha_0 + \sum_i \alpha_i \hat{x}_i \geq 0$.

We refer to such a game as *polynomially separable*. A direct extension of the proof of Theorem 1 implies that Nash equilibria can be found for polynomially separable games.

Theorem 4. *There is a polytime algorithm which finds a Nash Equilibrium of a given polynomially separable game.*

This general methodology can be used for finding a NE in many zero-sum games. In subsequent sections, we show how our framework can be used to find

Nash equilibria for a generalization of Blotto games, known as games, and for a class of dueling games introduced by Immorlica et al. [60].

We also show one can use similar techniques to compute the approximate equilibrium payoffs of a dueling game when we are not able to answer the separation problem in polynomial time but instead we can polynomially solve the ϵ-separation problem for any $\epsilon > 0$.

Theorem 5. *Given an oracle function for the ϵ-separation problem, one can find an ϵ-approximation to the equilibrium payoffs of a polynomially separable game in polynomial time.*

The General Lotto game is a relaxation of the Colonel Lotto game (See [36] for details). In this game each player's strategy is a distribution of a nonnegative integer-valued random variable with a given expectation. In particular, players A and B simultaneously determine (two distributions of) two nonnegative integer-valued random variables X and Y, respectively, such that $\mathbb{E}[X] = a$ and $\mathbb{E}[Y] = b$. The payoff of player A is

$$h_\Gamma^A(X, Y) = \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} \Pr(X = i) \Pr(Y = j) u(i, j), \tag{2.4}$$

and again the payoff of player B is the negative of the payoff of player A, i.e., $h_\Gamma^B(X, Y) = -h_\Gamma^A(X, Y)$. Hart [36] presents a solution for the General Lotto game when $u(i, j) = \text{sign}(i - j)$. Here, we generalize this result and present a polynomial-time algorithm for finding an equilibrium when u is a *bounded distance function*. Function u is a *bounded distance function*, if one can write it as $u(i, j) = f_u(i - j)$

such that f_u is a monotone function and reaches its maximum value at $u^M = f_u(u^T)$ where $u^T \in O(\text{poly}(a, b))$. Note that $u(i, j) = \text{sign}(i - j)$ is a bounded distance function where it reaches its maximum value at $i - j = 1$. Now, we are ready to present our main result regarding the General Lotto game.

Theorem 6. *There is a polynomial-time algorithm which finds an equilibrium of a General Lotto game where the payoff function is a bounded distance function.*

Main challenge Note that in the General Lotto game, each player has infinite number of pure strategies, and thus one cannot use neither our proposed algorithm for the Colonel Blotto game nor the technique of [60] for solving the problem. We should prune strategies such that the problem becomes tractable. Therefore, we characterize the extreme point of the polytope of all strategies, and use this characterization for pruning possible strategies.

To the best of our knowledge, our algorithm is the first algorithm of this kind which computes an NE of a game with infinite number of pure strategies.

2.1 Application to Dueling Games

Immorlica et al. [60] introduced the class of dueling games. In these games, an optimization problem with an element of uncertainty is considered as a competition between two players. They also provide a technique for finding Nash equilibria for a set of games in this class. Later Dehghani *et al.* [61] studied dueling games in a non-computational point of view and proved upper bounds on the price of anarchy of many dueling games. In this section, we formally define the dueling games and

bilinear duels. Then, in Section 2.1.3, we describe our method and show that our technique solves a more general class of dueling games. Furthermore, we provide examples to show how our method can be a simpler tool for solving bilinear duel games compared to [60] method. Finally, in Section 2.1.4, we examine the matching duel game to provide an example where the method of [60] does not work, but our presented method can yet be applied.

2.1.1 Dueling Games

Formally, dueling games are two player zero-sum games with a set of strategies X, a set of possible situations Ω, a probability distribution p over Ω, and a cost function $c : X \times \Omega \to \mathbb{R}$ that defines the cost measure for each player based on her strategy and the element of uncertainty. The payoff of each player is defined as the probability that she beats her opponent minus the probability that she is beaten. More precisely, the utility function is defined as

$$h^A(x, y) = -h^B(x, y) = \Pr_{\omega \sim p}[c(x, \omega) < c(y, \omega)] - \Pr_{\omega \sim p}[c(x, \omega) > c(y, \omega)]$$

where x and y are strategies for player A and B respectively. In the following there are two dueling games mentioned in [60].

Binary search tree duel. In the Binary search tree duel, there is a set of elements Ω and a probability distribution p over Ω. Each player is going to construct a binary search tree containing the elements of Ω. Strategy x beats strategy y for element $\omega \in \Omega$ if and only if the path from ω to the root in x is shorter than the

path from ω to the root in y. Thus, the set of strategies X is the set of all binary search trees with elements of Ω, and $c(x, \omega)$ is defined to be the depth of element ω in strategy x.

Ranking duel. In the Ranking duel, there is a set of m pages Ω, and a probability distribution p over Ω, notifying the probability that each page is going to be searched. In the Ranking duel, two search engines compete against each other. Each search engine has to provide a permutation of these pages, and a player beats the other if page ω comes earlier in her permutation. Hence, set of strategies X is all $m!$ permutations of the pages and for permutation $x = (x_1, x_2, \ldots, x_m)$ and page ω, $c(x, \omega) = i$ iff $\omega = x_i$.

2.1.2 Dueling Games are Polynomially Separable

Consider a dueling game in which each strategy $\hat{x}$ of player A is an $n(A)$ dimensional point in Euclidean space. Let S_A be the convex hull of these strategy points. Thus each point in S_A is a mixed strategy of player A. Similarly define strategy $\hat{y}$, $n(B)$, and S_B for player B. A dueling game is *bilinear* if utility function $h^A(\hat{x}, \hat{y})$ has the form $\hat{x}^t M \hat{y}$ where M is an $n(A) \times n(B)$ matrix. Again for player B, we have $h^B(\hat{x}, \hat{y}) = -h^A(\hat{x}, \hat{y})$. [60] provides a method for finding an equilibrium of a class of bilinear games which is defined as follows:

Definition 7. *Polynomially-representable bilinear dueling games: A bilinear dueling game is polynomially representable if one can present the convex hull of strategies S_A and S_B with m polynomial linear constraints, i.e. there are m vectors*

$\{v_1, v_2, \ldots, v_m\}$ *and* m *real numbers* $\{b_1, b_2, \ldots, b_m\}$ *such that* $S_A = \{\hat{x} \in R^{n(A)} | \forall i \in \{1, 2, \ldots, m\}, v_i$

$b_i\}$. *Similarly* $S_B = \{\hat{y} \in R^{n(B)} | \forall i \in \{1, 2, \ldots, m'\}, v'_i.\hat{y} \geq b'_i\}$.

In the following theorem, we show that every polynomially representable bilinear duel is also polynomially separable, as defined in Section 2. This implies that the general reduction described in Section 2 can be used to solve polynomially representable bilinear duels as well.

Theorem 8. *Every polynomially-representable bilinear duel is polynomially separable.*

Proof. Let S_A and S_B be the set of strategy points for player A and player B, respectively. We show that if S_A can be specified with polynomial number of linear constraints, then one could design an algorithm that finds out whether there exists a point $\hat{x} \in S_A$ such that $\alpha_0 + \sum \alpha_i \hat{x}_i \geq 0$. Let $\{(v_1, b_1), (v_2, b_2), \ldots, (v_m, b_m)\}$ be the set of constraints which specify S_A where v_i is a vector of size $n(A)$ and b_i is a real number. We need to check if there exists a point satisfying both constraints in $\{(v_1, b_1), (v_2, b_2), \ldots, (v_m, b_m)\}$ and $\alpha_0 + \sum \alpha_i \hat{x}_i \geq 0$. Recall that m is polynomial. Since all these constraints are linear, we can solve this feasibility problem by a LP in polynomial time. The same argument holds for S_B, therefore every polynomially representable bilinear duel is polynomially separable as well. □

2.1.3 A Simplified Argument for Ranking and Binary Search Duels

In this section, we revisit some examples of dueling games, and show how to use Theorem 4 to establish that they can be solved in polynomial time. The

application of Theorem 4 as two main steps. First, it is necessary to express the duel as a bilinear game: that is, one must transfer every strategy of the players to a point in $n(A)$ and $n(B)$ dimensional space, such the outcome of the game can be determined for two given strategy points with an $n(A) \times n(B)$ matrix M. Second, one must implement an oracle that determines whether there exists a strategy point satisfying a given linear constraint.

To illustrate our method more precisely, we propose a polynomial-time algorithm for finding an NE for ranking and binary search tree dueling games in what follows.

Theorem 9. *There exists an algorithm that finds an NE of the Ranking duel in polynomial time.*

Proof. We transfer each strategy x of player A to point $\hat{x}$ in $\mathbb{R}^{m^2}$ where $\hat{x}_{i,j}$ denotes the probability that ω_i stands at position j in x. The outcome of the game is determined by the following equation

$$\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=j+1}^{m}\hat{x}_{i,j}\hat{y}_{i,k}p(\omega_i) - \sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{j-1}\hat{x}_{i,j}\hat{y}_{i,k}p(\omega_i)$$

Where $p(\omega_i)$ denotes the probability that ω_i is searched.

Here, we need to provide an oracle that determines whether there exists a strategy point for a player that satisfies a given linear constraint $\alpha_0 + \sum \alpha_{i,j}\hat{x}_{i,j} \geq 0$. Since each pure strategy is a matching between pages and indices, we can find the pure strategy that maximizes $\sum \alpha_{i,j}\hat{x}_{i,j}$ with the maximum weighted matching algorithm. Therefore, this query can be answered in polynomial time. Since we

have reduced this game to a polynomially-separable bilinear duel, we can find a Nash equilibrium in polynomial time. □

Theorem 10. *There exists an algorithm that finds an NE of the Binary search tree duel in polynomial time.*

Proof. Here we map each strategy x to the point $\hat{x} = \langle \hat{x}_{1,1}, \hat{x}_{1,2}, \ldots, \hat{x}_{1,m}, \hat{x}_{2,1}, \hat{x}_{2,2}, \ldots, \hat{x}_{m,m} \rangle \in \mathbb{R}^{m^2}$ where $\hat{x}_{i,j}$ denotes the probability that depth of the i-th element is equal to j. Therefore, the payoff of the game for strategies $\hat{x}$ and $\hat{y}$ is equal to

$$\sum_{i=1}^{m} \sum_{j=1}^{m} \sum_{k=j+1}^{m} \hat{x}_{i,j} \hat{y}_{i,k} p(\omega_i) - \sum_{i=1}^{m} \sum_{j=1}^{m} \sum_{k=1}^{j-1} \hat{x}_{i,j} \hat{y}_{i,k} p(\omega_i)$$

Where $p(\omega_i)$ denotes the probability that i-th element is searched.

Next, we need to provide an oracle that determines whether there exists a strategy point for a player that satisfies a given linear constraint $\alpha_0 + \sum \alpha_{i,j} \hat{x}_{i,j} \geq 0$. To do this, we find the binary search tree that maximizes $\sum \alpha_{i,j} \hat{x}_{i,j}$. This can be done with a dynamic program. Let $D(a, b, k)$ denote the maximum value of $\sum_{i=a}^{b} \alpha_{i,j} \hat{x}_{i,j}$ for a subtree that its root is at depth k. $D(a, b, k)$ can be formulated as

$$D(a,b,k) = \begin{array}{l} a < b,\ \min_{a \leq c \leq b} \{D(a, c-1, k+1) + D(c+1, b, k+1) + \alpha_{c,k}\} \\ \\ a = b,\ \alpha_{a,k} \end{array}$$

Therefore, we can find the Binary search tree which maximizes $\sum \alpha_{i,j} \hat{x}_{i,j}$ in poly-

nomial time and see if it meets the constraint. Since we have reduced this game to a polynomially-separable bilinear duel, we can find an NE in polynomial time. □

2.1.4 Matching Duel

In matching duel we are given a weighted graph $G = (V, E, W)$ which is not necessarily bipartite. In a matching duel each pure strategy of players is a perfect matching, set of possible situations Ω is the same as the set of nodes in G, and probability distribution p over Ω determines the probability of selection of each node. In this game, strategy x beats strategy y for element $\omega \in \Omega$ if ω is matched to a higher weighted edge in strategy x than strategy y.

The matching duel may find its application in a competition between websites that try to match people according to their desire. In this competition the website that suggest a better match for each user will get that user, and the goal of each website is to maximize the number of its users. We mention that the ranking duel is a special case of the matching duel, when G is a complete bipartite graph with n nodes on each side, in which the first part denotes the web pages and the second part denotes the positions in the ranking. Thus, the weight of the edge between page i and rank j is equal to j.

First, we describe how our method can solve this game and then we show the method of Immorlica et al. [60] cannot be applied to find an NE of the matching duel.

Theorem 11. *There exists an algorithm that finds an NE of the matching duel in*

polynomial time.

Proof. We transfer every strategy x to a point in $|E|$-dimensional Euclidean space $\hat{x}$, where $\hat{x}_e$ denotes the probability that x chooses e in the matching. Thus, the payoff function is bilinear and is as follows:

$$\sum_{\omega\in\Omega}\sum_{e_1\in N(\omega)}\sum_{e_2\in N(\omega)}[p(\omega)\hat{x}_{e_1}\hat{y}_{e_2}\times \text{sign}(w(e_1)-w(e_2))]$$

where $N(\omega)$ is the set of edges adjacent to ω[2]. Next, we need to prove that the game is polynomially separable. That is, given a vector α and a real number α_0, we are to find out whether there is a strategy $\hat{x}$ such that $\alpha_0+\alpha.\hat{x}\geq 0$. This problem can be solved by a maximum weighted prefect matching, where the graph is $G=(V,E,W)$ and $w(e)=\alpha_e$. Thus our framework can be used to find an NE of the matching duel in polynomial time. □

Note that Rothvoss [62] showed that the feasible strategy polytope (the perfect matching polytope) has exponentially many facets. Therefore, the prior approach represented in the work of Immorlica et al. [60] is not applicable to the matching duel. This example shows that our framework nontrivially generalizes the method of Immorlica et al. [60] and completes the presentation of our simpler and more powerful tool for solving bilinear duels.

[2]Note that sign(w) is 1, -1, and 0 if w is positive, negative, and zero respectively.

2.2 Colonel Blotto

In this section we provide a more detailed description of our polynomial time algorithm for finding a Nash equilibrium (NE) of the Colonel Blotto game. Hart showed that the Colonel Lotto game is equivalent to a special case of the Colonel Blotto game [36]. Therefore, our algorithm could be used to find a NE of the Colonel Lotto game as well.

In section 2.2.1, we present a procedure for mapping strategies of both players to a new space. The new space maintains the important information of each strategy and helps us to find a Nash equilibrium of the game. Next in section 2.2.2, we show how we check the feasibility of the membership constraint in the new space. Moreover, we present a polynomial-time algorithm for determining an equilibrium of the Colonel Blotto game in the new space. At last in section 2.2.3, we present an algorithm which transfers a Nash equilibrium from the new space to the original space.

2.2.1 Transferring to a New Space

In this subsection we define a new notation for describing the strategies of players and discuss about the properties of the transferred strategies. Let $n(A) = k(a+1)$, and $\mathbf{x}$ be a strategy of player A. We define the function $\mathcal{G}_A$ in the following way: $\mathcal{G}_A(\mathbf{x}) = \hat{\mathbf{x}}$ where $\hat{\mathbf{x}}$ is a point in $\mathbb{R}^{n(A)}$ such that $\hat{x}_{(i-1)(a+1)+j+1}$ is equal to the probability that strategy $\mathbf{x}$ puts j units in the i-th battlefield, for $1 \leq i \leq k$ and $0 \leq j \leq a$. For simplicity we may represent $\hat{x}_{(i-1)(a+1)+j+1}$ by $\hat{x}_{i,j}$. We define $n(B)$

and $\mathcal{G}_B$ similarly for player B. Let $n = max\{n(A), n(B)\}$. Note that, $\mathcal{G}_A$ maps each strategy of the first player to exactly one point in $\mathbb{R}^{n(A)}$. However, each point in $\mathbb{R}^{n(A)}$ may be mapped to zero, one, or more than one strategies. Let us recall the definition of $\mathcal{M}(\mathcal{X})$ which is the set of all strategies of player A, and the definition of S_A which is

$$S_A = \{\hat{\mathbf{x}} \in [0,1]^{n(A)} | \exists \mathbf{x} \in \mathcal{M}(\mathcal{X}), \mathcal{G}_A(\mathbf{x}) = \hat{\mathbf{x}}\}.$$

In order to design an algorithm for checking the membership constraint, we first demonstrate in Lemma 12 that set S_A is a polyhedron with an exponential number of vertices and facets. This lemma is a more formal statement of Lemma 2. Then we prove in Lemma 13 that set S_A can be formulated with $\mathcal{O}(2^{\text{poly}(n)})$ number of constraints. These results allow us to leverage the Ellipsoid method for checking the membership constraint [57].

Lemma 12. *Set S_A forms a convex polyhedron with no more than $n(A)^{n(A)}$ vertices and no more than $n(A)^{(n(A)^2)}$ facets.*

Lemma 13. *Set S_A can be formulated with $\mathcal{O}(2^{\text{poly}(n)})$ number of constraints*

2.2.2 Verifying the Membership Constraints and the Payoff Constraints

The final goal of this section is to determine a NE of the Colonel Blotto game. To do this, we provide linear program 2.2 and show that this LP can be solved in polynomial time. Since we use the Ellipsoid method to solve the LP, we

have to implement an oracle function that reports a violating constraint for any infeasible solution. In this subsection we focus on the membership constraint of LP 2.2 and show that for any infeasible point $\hat{\mathbf{x}}$ which violates membership constraint, a polynomial-time algorithm finds a hyperplane that separates $\hat{\mathbf{x}}$ from S_A.

Lemma 14. *There exists a polynomial time algorithm that gets a point $\hat{\mathbf{x}}$ as input, and either finds a hyperplane that separates $\hat{\mathbf{x}}$ from S_A, or reports that no such hyperplane exists.*

Proof. Let $\hat{\mathbf{x}} = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_{n(A)})$. Consider the following LP, which we will refer to as LP 2.5:

$$\alpha_0 + \sum_{j=1}^{n(A)} \alpha_j \hat{x}_j < 0 \tag{2.5}$$

$$\alpha_0 + \sum_{j=1}^{n(A)} \alpha_j \hat{v_j} \geq 0 \qquad \forall \hat{v} \in I_A \tag{2.6}$$

The variables of the linear program are $\alpha_0, \alpha_1, \ldots, \alpha_{n(A)}$, which describe the following hyperplane:

$$\alpha_0 + \sum_{j=1}^{n(A)} \alpha_j \hat{x}'_j = 0.$$

Constraints 2.5 and 2.6 force LP 2.5 to find a hyperplane that separates $\hat{\mathbf{x}}$ from S_A. Hence, LP 2.5 finds a separating hyperplane if and only if $\hat{\mathbf{x}}$ is not in S_A.

Hyperplane separating oracle is an oracle that gets variables $\alpha_0, \alpha_1, \ldots, \alpha_{n(A)}$ as input and finds if constraints 2.6 are satisfied. Moreover, if some constraints are violated it returns at least one of the violated constraints. In Section 2.4 we describe a polynomial-time algorithm for the hyperplane separating oracle. Constraint 2.5

also can be checked in polynomial time. Our LP has $n(A)+1$ variables and $|I_A|+1$ constraints which is $\mathcal{O}(2^{\text{poly}(n)})$ by Lemma 12. Thus we can solve this LP in polynomial time with the Ellipsoid method [57]. □

In the next step, we present an algorithm to determine the outcome of the game when both players play optimally. We say $\mathbf{x}$ is an optimal strategy of player A, if it maximizes the guaranteed payoff of player A. By the MinMax Theorem, in a NE of a zero-sum game players play optimally [55]. Therefore, it is enough to find an optimal strategy of both players. Before we discuss the algorithm, we show the payoff $h^A_{\mathcal{B}}(\mathbf{x},\mathbf{y})$ can be determined by $\mathcal{G}_A(\mathbf{x})$ and $\mathcal{G}_B(\mathbf{y})$. Recall the definition of $h^A_{\mathcal{B}}(\hat{\mathbf{x}},\hat{\mathbf{y}})$ which is

$$h^A_{\mathcal{B}}(\hat{\mathbf{x}},\hat{\mathbf{y}}) = \sum_{i=1}^{k}\sum_{\alpha=0}^{a}\sum_{\beta=0}^{b}\hat{x}_{i,\alpha}\hat{y}_{i,\beta}u_i^A(\alpha,\beta).$$

Lemma 15. *Let $\mathbf{x}\in\mathcal{M}(\mathcal{X})$ and $\mathbf{y}\in\mathcal{M}(\mathcal{Y})$ be two mixed strategies for player A and B respectively. Let $\hat{\mathbf{x}}=\mathcal{G}_A(\mathbf{x})$ and $\hat{\mathbf{y}}=\mathcal{G}_B(\mathbf{y})$. The outcome of the game is determined by $h^A_{\mathcal{B}}(\hat{\mathbf{x}},\hat{\mathbf{y}})$.*

Proof. Let $\mathbf{x}$ and $\mathbf{y}$ be two mixed strategies of players A and B respectively, and let $\mathbb{E}[u_i^A(\mathbf{x},\mathbf{y})]$ be the expected value of the outcome in battlefield i. We can write $\mathbb{E}[u_i^A(\mathbf{x},\mathbf{y})]$ as follows

$$\mathbb{E}[u_i^A(\mathbf{x},\mathbf{y})] = \sum_{\alpha=0}^{a}\sum_{\beta=0}^{b}\hat{x}_{i,\alpha}\hat{y}_{i,\beta}u_i^A(\alpha,\beta).$$

We know that the total outcome of the game is the sum of the outcome in all

battlefields, which is

$$\mathbb{E}\left[\sum_{i=1}^{k} u_i^A(\mathbf{x},\mathbf{y})\right] = \sum_{i=1}^{k} \mathbb{E}[u_i^A(\mathbf{x},\mathbf{y})] = \sum_{i=1}^{k}\sum_{\alpha=0}^{a}\sum_{\beta=0}^{b} \hat{x}_{i,\alpha}\hat{y}_{i,\beta} u_i^A(\alpha,\beta) = h_{\mathcal{B}}^A(\hat{\mathbf{x}},\hat{\mathbf{y}}).$$

□

Theorem 16. *There exists a polynomial time algorithm that finds a NE of the Colonel Blotto game in the new space.*

Proof. The Colonel Blotto is a zero-sum game, and the MinMax theorem states that a pair of strategies $(\hat{\mathbf{x}},\hat{\mathbf{y}})$ is a Nash equilibrium if $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ maximize the guaranteed payoff of players A and B respectively [55].

Recall that LP 2.2 finds a point $\hat{\mathbf{x}} \in S_A$ which describes an optimal strategy of player A[3]. This LP has $n(A)+1$ variables $\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_{n(A)}$ and U where $\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_{n(A)}$ describe point $\hat{\mathbf{x}}$. The membership constraint guarantees $\hat{\mathbf{x}}$ is in S_A. It is known that in any normal-form game there always exists a best-response strategy which is a pure strategy [63]. Hence, variable U represents the maximum payoff of player A with strategy $\hat{\mathbf{x}}$ when player B plays his best-response strategy against $\hat{\mathbf{x}}$. Note that, Lemma 15 shows $h_{\mathcal{B}}^A(\hat{\mathbf{x}},\hat{\mathbf{y}})$ is a linear function of $\hat{\mathbf{x}}$, when $\hat{\mathbf{y}}$ is a fixed strategy of player B. This means the payoff constraints are linear constraints. Putting all these together show, LP 2.2 finds a point $\hat{\mathbf{x}}$ such that:

1. There exists strategy $\mathbf{x}$ such that $\mathcal{G}_A(\mathbf{x}) = \hat{\mathbf{x}}$.

2. The minimum value of $h_{\mathcal{B}}^A(\mathbf{x},\mathbf{y})$ is maximized for every $\mathbf{y} \in \mathcal{M}(\mathcal{Y})$.

[3]The same procedure finds an optimal strategy of player B.

Next we show that this LP can be solved in polynomial time with the Ellipsoid method. First, Lemma 14 proposes a polynomial-time algorithm for checking the membership constraint. Second, *best-response separating oracle* is an oracle that gets point $\hat{\mathbf{x}}$ and variable U as input and either reports point $\hat{\mathbf{x}}$ passes all payoff constrains or reports a violated payoff constraint. In Section 2.4, we will show that the running time of this oracle is $\mathcal{O}(\text{poly}(n))$.

At last we prove LP 2.2 has $\mathcal{O}(2^{\text{poly}(n)})$ number of constraints, and we can leverage the Ellipsoid method for finding a solution of this LP. Note that Lemma 13 indicates that set S_A can be represented by $\mathcal{O}(2^{\text{poly}(n)})$ number of hyperplanes. On the other hand, Lemma 12 shows LP 2.2 has at most $|I_B| = \mathcal{O}(2^{\text{poly}(n)})$ constraints.

□

2.2.3 Finding a Nash Equilibrium in the Original Space

In the previous subsection, we presented an algorithm which finds a Nash equilibrium $(\mathcal{G}_A(\mathbf{x}), \mathcal{G}_B(\mathbf{y}))$ of the game in the new space. The remaining problem is to retrieve $\mathbf{x}$ from $\mathcal{G}_A(\mathbf{x})$.

Theorem 17. *Given a point $\hat{\mathbf{x}} \in S_A$, there exists a polynomial time algorithm which finds a strategy $\mathbf{x} \in \mathcal{M}(\mathcal{X})$ such that $\mathcal{G}_A(\mathbf{x}) = \hat{\mathbf{x}}$.*

Proof. Since every strategy of player A is a convex combination of elements of $\mathcal{X}$,

our goal is to find a feasible solution of the following LP.

$$\min . \quad 0 \tag{2.7}$$

$$s.t. \quad \sum_{x \in \mathcal{X}} \alpha_x = 1 \tag{2.8}$$

$$\sum_{x \in \mathcal{X}} \alpha_x \mathcal{G}_A(x)_j = \hat{x}_j \qquad \forall 1 \leq j \leq n(A) \tag{2.9}$$

$$\alpha_x \geq 0 \qquad \forall x \in \mathcal{X} \tag{2.10}$$

where α_x is the probability of pure strategy $x \in \mathcal{X}$. Note that, this LP finds a mixed strategy of player A by finding the probability of each pure strategy. Since every feasible solution is acceptable, objective function does not matter. To find a solution of this LP, we write its dual LP as follows.

$$\max . \quad \beta_0 + \sum_{j=1}^{n(A)} \hat{x}_j \beta_j \tag{2.11}$$

$$s.t. \quad \beta_0 + \sum_{j=1}^{n(A)} \mathcal{G}_A(x)_j \beta_j \leq 0 \qquad \forall x \in \mathcal{X} \tag{2.12}$$

Where variable β_0 stands for constraint 2.8, and variables $\beta_1, \beta_2, \ldots, \beta_{n(A)}$ stand for constraints 2.9. An oracle similar to the *hyperplane separating oracle* can find a violating constraint for any infeasible solution of the dual LP. Since the number of constraints in the dual LP is $|I_A| = \mathcal{O}(2^{\text{poly}(n)})$ based on Lemma 12, we can use the Ellipsoid method and find an optimal solution of the dual LP in polynomial time.

The next challenge is to find an optimal solution of the primal LP from an

optimal solution of the dual LP. We resolve this problem by the following lemma. We know $\hat{\mathbf{x}}$ is in S_A. This means there is strategy $\mathbf{x} \in \mathcal{M}(\mathcal{X})$ such that $\mathcal{G}_A(\mathbf{x}) = \hat{\mathbf{x}}$. Hence, linear program 2.7 and its dual are feasible, and we can apply Lemma 18. $\square$

Lemma 18. *Assume we have a separation oracle for primal LP* $\max\{c^T x : Ax \leq b\}$ *with exponentially many constraints and polynomially many variables. If primal LP is feasible, then there is a polynomial-time algorithm which returns an optimum solution of dual LP* $\min\{b^T y : A^T y \geq c\}$.

Proof. Since the primal LP is feasible, we can assume $OPT = \max\{c^T x : Ax \leq b\}$. The Ellipsoid method returns an optimum solution of primal LP by doing binary search and finding the largest K which guarantees feasibility of $\{c^T x \leq K : Ax \leq b\}$. Let $(\hat{A}, \hat{b})$ be the set of polynomially many constraints returned by the separation oracle during all iterations. We first prove $\max\{c^T x : \hat{A}x \leq \hat{b}\} = OPT$. Note that $(\hat{A}, \hat{b})$ is a set of constraints returned by the Ellipsoid method. Note that $(\hat{A}, \hat{b})$ is a subset of all constraints (A, b). This means every vector x which satisfies $Ax \leq b$ will satisfy $\hat{A}x \leq \hat{b}$ as well. Therefore, $\max\{c^T x : \hat{A}x \leq \hat{b}\} \geq \max\{c^T x : Ax \leq b\} = OPT$. On the other hand, we know $(\hat{A}, \hat{b})$ contains constraints which guarantees infeasibility of $\max\{c^T x \geq OPT + \epsilon : Ax \leq b\}$. So, LP $\max\{c^T x \geq OPI + \epsilon : \hat{A}x \leq \hat{b}\}$ is infeasible which means $\max\{c^T x : \hat{A}x \leq \hat{b}\} \leq OPT$. Putting all these together we can conclude that $\max\{c^T x : \hat{A}x \leq \hat{b}\} = OPT$.

Linear program $\max\{c^T x : \hat{A}x \leq \hat{b}\}$ has polynomially many constraints and polynomially many variables, and we can find an optimum solution to its dual $\min\{\hat{b}^T \hat{y} : \hat{A}^T \hat{y} \geq c\}$ in polynomial time. Let $\hat{y}^*$ be an optimum solution of dual LP

$\min\{\hat{b}^T\hat{y} : \hat{A}^T\hat{y} \geq c\}$, and let $S = \{i|(A_i, b_i) \text{ is in } (\hat{A}, \hat{b})\}$ where A_i is the i-th row of matrix A, i.e., be set of indices corresponding to constraints in $(\hat{A}, \hat{b})$. For every vector y and every set of indices R we define y_R to be the projection of vector y on set R. Now consider vector y^* as a solution of dual LP $\min\{b^T y : A^T y \geq c\}$ such that $y^*_S = \hat{y}^*$ and $y^*_i = 0$ for all $i \notin S$. We prove y^* is an optimum solution of dual LP $\min\{b^T y : A^T y \geq c\}$ as follows:

- We first show y^* is feasible. Note that $y^*_i = 0$ for all $i \notin S$ which means $A^T y^* = \hat{A}^T \hat{y}^* \geq c$ where the last inequality comes from the feasibility of $\hat{y}^*$ in dual LP $\min\{\hat{b}^T y : \hat{A}^T y \geq c\}$.

- Note that $b^T y^* = \sum_i b_i^T y_i^* = \sum_{i \in S} b_i^T y_i^* + \sum_{i \notin S} b_i^T y_i^* = \hat{b}^T \hat{y}^*$. The last equality comes from the facts that $y_i^* = 0$ for all $i \notin S$, and $\sum_{i \in S} b_i^T y_i^* = \hat{b}^T \hat{y}^*$. Since $\hat{y}^*$ is an optimum solution of dual the LP $\min\{\hat{b}^T y : \hat{A}^T y \geq c\}$, by the weak duality, it is equal to $\max\{c^T x : \hat{A}x \leq \hat{b}\} = OPT$. Therefore, $b^T y^* = OPT$.

We have proved y^* is a feasible solution to dual LP $\min\{b^T y : A^T y \geq c\}$ and $b^T y^* = OPT$. We also know $OPT = \max\{c^T x : Ax \leq b\}$ by definition. Therefore, the weak duality insures y^* is an optimum solution of dual LP $\min\{b^T y : A^T y \geq c\}$.

□

2.3 General Lotto

In this section we study the General Lotto game. An instance of the General Lotto game is defined by $\Gamma(a, b, u)$, where players A and B simultaneously define probability distributions of non-negative integers X and Y, respectively, such that

$\mathbb{E}[X] = a$ and $\mathbb{E}[Y] = b$. In this game, player A's aim is to maximize $h^A_\Gamma(X, Y)$ and player B's aim is to maximize $h^B_\Gamma(X, Y) = -h^A_\Gamma(X, Y)$ where $h^A_\Gamma(X, Y)$ is defined as $\mathbb{E}_{i\sim X, j\sim Y} u(i, j)$. The previous studies of the General Lotto game considered a special case of the problem where $u(i, j) = sign(i - j)$[4]. Here, we generalize the payoff function to a bounded distance function and present an algorithm for finding a Nash equilibrium of the General Lotto game in this case. Function u is a bounded distance function, if one can write it as $u(i, j) = f_u(i-j)$ such that f_u is a monotone function and reaches its maximum value at $f_u(t_u)$ where $t_u \in O(\text{poly}(a, b))$.

We first define a new version of the General Lotto game, which is called the *finite General Lotto* game. We prove a Nash equilibrium of the finite General Lotto game can be found in polynomial time. Then we reduce the problem of finding a Nash equilibrium of the General Lotto game with a bounded distance function to the problem of finding a Nash equilibrium of the finite General Lotto game. This helps us to propose a polynomial-time algorithm which finds a Nash equilibrium of the General Lotto game where the payoff function is a bounded distance function.

2.3.1 Finite General Lotto

We define the finite General Lotto game $\Gamma(a, b, u, S)$ to be an instance of the General Lotto game where every strategy of players is a distribution over a finite set of numbers S. Here, we leverage our general technique to show the finite General Lotto game is a polynomially-separable bilinear game and, as a consequence, it leads

[4] $sign(x) = \begin{cases} -1 & x < 0 \\ 0 & x = 0 \\ 1 & x > 0 \end{cases}$

to a polynomial time algorithm to find a Nash equilibrium for this game.

Theorem 19. *There exists an algorithm which finds a Nash equilibrium of the finite General Lotto game* $\Gamma(a, b, u, S)$ *in time* $O(poly(|S|))$.

Proof. First we map each strategy X to a point $\hat{x} = \langle \hat{x}_1, \hat{x}_2, \ldots, \hat{x}_{|S|} \rangle$, where $\hat{x}_i$ denotes $\Pr(X = S_i)$. Without loss of generality we assume the elements of S are sorted in strictly ascending order, i.e. for each $1 \leq i < j \leq |S|$, $S_i < S_j$. Now the utility of player A when A plays a strategy corresponding to $\hat{x}$ and B plays a strategy corresponding to $\hat{y}$ is obtained by the following linear function.

$$h_\Gamma^A(\hat{x}, \hat{y}) = \sum_{i=1}^{|S|} \sum_{j=1}^{i-1} \hat{x}\hat{y}u(i,j) - \sum_{i=1}^{|S|} \sum_{j=i+1}^{|S|} \hat{x}\hat{y}u(i,j).$$

Therefore the game is bilinear. Now we prove the game is polynomially separable.

Given a real number r and a vector v, we provide a polynomial-time algorithm which determines whether there exists a strategy point $\hat{x}$ such that $r + v.\hat{x} \geq 0$. We design the following feasibility program for this problem with $|S|$ variables $\hat{x}_1$ to $\hat{x}_{|S|}$ and three constraints.

$$\sum_{i=1}^{|S|} \hat{x}_i = 1 \tag{2.13}$$

$$\sum_{i=1}^{|S|} \hat{x}_i S_i = a \tag{2.14}$$

$$r + v.\hat{x} \geq 0 \tag{2.15}$$

Constraints 2.13 and 2.14 force the variables to represent a valid strategy point (i.e.,

the probabilities sum to 1 and the expectation equals a). Thus every point $\hat{x}$ is a valid strategy point *iff* it satisfies Constraints 2.13 and 2.14. On the other hand, Constraint 2.15 enforces the program to satisfy the given linear constraint of the separation problem. Thus there exists a strategy point $\hat{x}$ such that $b + v.\hat{x} \geq 0$ *iff* there is a solution for the feasibility LP. The feasibility of the program can be determined in polynomial time, hence, the separation problem is polynomially tractable.

Therefore the finite General Lotto game is a polynomially-separable bilinear game and by Theorem 4, there exists a polynomial-time algorithm which finds a Nash equilibrium of the finite General Lotto game. □

2.3.2 General Lotto with Bounded Distance Functions

In this section, we consider General Lotto game $\Gamma(a, b, u)$ where u is a bounded distance function and design a polynomial-time algorithm for finding a Nash equilibrium of the game. In the following part of this section we assume $a \leq b$. Recall the definition of bounded distance functions.

Definition 20. *Function u is a bounded distance function, if one can write it as $u(i, j) = f_u(i - j)$ such that f_u is a monotone function and reaches its maximum value at $u^M = f_u(u^T)$ where $u^T \in O(\text{poly}(a, b))$. We call u^T the threshold of function u, and u^M the maximum of function u* [5] .

First we define a notion of paired strategies and claim that for every strategy

[5]Note that sign function is a special case of distance functions.

of a player there is a best-response strategy which is a paired strategy. Then, using this observation, we can prove nice bounds on the set of optimal strategies.

Consider a probability distribution which only allows two possible outcomes, i.e., there are only two elements in S with non-zero probabilities. We call such a distribution a *paired strategy*. We define $T_{i,j}$ to be a paired strategy which only has non-zero probabilities at elements i and j. Furthermore, we define $T^a_{i,j}$ to be a paired strategy with $\mathbb{E}[T^a_{i,j}] = a$. In paired strategy $T^a_{i,j}$, the probabilities of elements i and j are determined by $\alpha^a_{i,j} = Pr(T^a_{i,j} = i) = \frac{a-j}{i-j}$ and $\alpha^a_{j,i} = Pr(T^a_{i,j} = j) = \frac{a-i}{j-i} = 1-\alpha^a_{i,j}$. In the following structural lemma, we show that every distribution T over a finite set S can be constructed by a set of paired strategies.

Lemma 21. *For every distribution T over S with $\mathbb{E}[T] = a$ and t elements with non-zero probability, there are $m \leq t$ paired strategies $\sigma_1, \sigma_2, \ldots, \sigma_m$ such that $T = \sum_{r=1}^{m} \beta_r {\sigma_r}$[6], and for all $1 \leq i \leq m$ we have $\beta_i \in [0,1]$ and $\mathbb{E}[\sigma_i] = a$.*

Proof. We prove this claim by induction on the number of elements with non-zero probabilities in T. If there is only one element with non-zero probability in T, i.e., $t = 1$, then we have $Pr(T = a) = 1$. Thus $T = T^a_{0,a}$ is a paired strategy and the claim holds by setting $\sigma_1 = T^a_{0,a}$ and $\beta_1 = 1$. Now assuming the claim holds for all $1 \leq t' < t$, we prove the claim also holds for t. Suppose T be a probability distribution with t non-zero probability elements. Since $t \geq 2$, there should be some $i < a$ with $Pr(T = i) > 0$ and some $j > a$ with $Pr(T = j) > 0$. We choose the largest possible number $0 < \beta \leq 1$ such that $\beta\alpha^a_{i,j} \leq Pr(T = i)$ and

[6]This lemma claims that the strategy of a player in the finite General Lotto game can be written as a probability distribution over paired strategies. Thus $T = \beta_1\sigma_1 + \beta_2\sigma_2 + \ldots + \beta_m\sigma_m$ describes a strategy in which the paired strategy σ_i is played with probability β_i.

$\beta\alpha^a_{j,i} \leq Pr(T = j)$. If $\beta = 1$, then $Pr(T = i) + Pr(T = j) \geq \alpha^a_{i,j} + \alpha^a_{j,i} = 1$. This means $T = T^a_{i,j}$ is a paired strategy and the claim holds by setting $\sigma_1 = T^a_{i,j}$ and $\beta_1 = 1$. Otherwise, we can write $T = (1 - \beta)T' + \beta T^a_{i,j}$ where $T' = \frac{T - \beta T^a_{i,j}}{1-\beta}$. Furthermore we have

$$\mathbb{E}[T'] = \frac{\mathbb{E}[T - \beta T^a_{i,j}]}{1 - \beta} = \frac{\mathbb{E}[T] - \beta E[T^a_{i,j}]}{1 - \beta} = a.$$

We select β such that at least one of the probabilities $Pr[T' = i]$ or $Pr[T' = j]$ becomes zero. Thus compared to T, the number of elements with non-zero probability in T' is at least decreased by one, and by the induction hypothesis we can write $T' = \beta'_1\sigma'_1 + \beta'_2\sigma'_2 + \ldots + \beta'_{m'}\sigma'_{m'}$ where $m' \leq t - 1$. Let $\beta_i = (1 - \beta)\beta'_i$ and $\sigma_i = \sigma'_i$ for $1 \leq i \leq m'$ and $\beta_{m'+1} = \beta$ and $\sigma_{m'+1} = T^a_{i,j}$. Now we can write $T = \beta_1\sigma_1 + \beta_2\sigma_2 + \ldots + \beta_{m'+1}\sigma_{m'+1}$ where each σ_i is a paired strategy and $E(\sigma_i) = a$. Furthermore, $m = m' + 1 \leq t$ and the proof is complete. Since $t \leq |S|$, m is polynomial in the size of input. Therefore paired strategies $\sigma_1, \sigma_2, \ldots, \sigma_m$ and their corresponding coefficients $\beta_1, \beta_2, \ldots, \beta_m$ can be computed in polynomial time. □

Lemma 22. *For every strategy of player A in a finite General Lotto game there is a best-response strategy of player B which is a paired strategy.*

Proof. Consider finite General Lotto game $\Gamma(a, b, u, S)$, strategy X of player A, and a best-response strategy Z of player B. Since Z is a distribution on S, by using Lemma 21 we can write $Z = \sum_{r=1}^{m} \beta_r\sigma_r$. Thus, we have $h^B_\Gamma(X, Z) = h^B_\Gamma(X, \sum_{r=1}^{m} \beta_r\sigma_r)$ and because of the linearity of expectation we can write $h^B_\Gamma(X, Z) = \sum_{r=1}^{m} \beta_r h^B_\Gamma(X, \sigma_r)$.

Since Z is a best-response strategy, we have:

$$\forall 1 \leq r \leq m, \;\; h_{\Gamma}^{B}(X, \sigma_r) = h_{\Gamma}^{B}(X, Z). \tag{2.16}$$

This means paired strategy σ_r, for each $1 \leq r \leq m$, is a best-response strategy of player B. □

In the following lemmas, using the structural property of the best-response strategies, we show some bounds for each player's optimal strategies.

Lemma 23. *For any strategy X with $\mathbb{E}[X] = c$ and any integer j we have $\sum_{i=0}^{j} Pr(X = i) \geq 1 - \frac{c}{j+1}$.*

Proof. Since $\sum_{i=0}^{+\infty} i\Pr(X = i) = c$, we have $(j+1)\sum_{i=j+1}^{+\infty} \Pr(X = i) \leq c$. This implies

$$\sum_{i=0}^{j} \Pr(X = i) = 1 - \sum_{i=j+1}^{+\infty} \Pr(X = i) \geq 1 - \frac{c}{j+1}.$$

□

Lemma 24. *Consider Nash equilibrium (X, Y) of General Lotto game $\Gamma(a, b, u)$ where u is a bounded distance function with threshold u^T. We have $\sum_{i=0}^{a-1} Pr(Y = i) \leq \frac{u^T}{u^T+1}$.*

Proof. Let X' be a pair distribution of player A that chooses $a-1$ with probability p and chooses $a + u^T - 1$ with probability $1 - p$. Thus $p = \frac{u^T-1}{u^T}$. The payoff of

playing strategy X' against Y is

$$h_{\Gamma}^{A}(X', Y) = \sum_{i=0}^{+\infty} \Pr(Y = i)[pu(a-1, i) + (1-p)u(a+u^{T}-1, i)]. \tag{2.17}$$

Note that by the definition of u, $u(i,j) \geq 0$ if and only if $i - j \geq 0$ and $u(i,j) \leq 0$ if and only if $i - j \leq 0$. Furthermore, if $i - j \geq u^{T}$ then $u(i,j) = u^{M}$ and if $i - j \leq -u^{T}$ then $u(i,j) = -u^{M}$. Therefore,

$$\begin{aligned}
\sum_{i=0}^{a-1} \Pr(Y = i) pu(a-1, i) &\geq 0 \\
\sum_{i=0}^{a-1} \Pr(Y = i)(1-p)u(a+u^{T}-1, i) &\geq (1-p)u^{M} \sum_{i=0}^{a-1} \Pr(Y = i) \\
\sum_{i=a}^{+\infty} \Pr(Y = i)[pu(a-1, i) + (1-p)u(a+u^{T}-1, i)] &\geq -u^{M} \sum_{i=a}^{+\infty} \Pr(Y = i)
\end{aligned} \tag{2.18}$$

Note that $a \leq b$ and (X, Y) is a Nash equilibrium which means $h_{\Gamma}^{A}(X, Y) \leq 0$. This implies $h_{\Gamma}^{A}(X', Y) \leq 0$. Thus, by applying Equality 2.17 and Inequality 2.18 we have

$$0 \geq h_{\Gamma}^{A}(X', Y) \geq (1-p)u^{M} \sum_{i=0}^{a-1} \Pr(Y = i) - u^{M} \sum_{i=a}^{+\infty} \Pr(Y = i),$$

which implies $\sum_{i=0}^{+\infty} \Pr(Y = i) \geq (2-p) \sum_{i=0}^{a-1} \Pr(Y = i)$. Thus, $\sum_{i=0}^{a-1} \Pr(Y = i) \leq \frac{1}{2-p}$. By substituting $\frac{u^{T}-1}{u^{T}}$ instead of p we can conclude $\sum_{i=0}^{a-1} \Pr(Y = i) \leq \frac{u^{T}}{u^{T}+1}$. $\square$

In the following lemma we provide an upper-bound for the maximum variable with non-zero probability of a player's strategy in the equilibrium.

Lemma 25. *Consider a Nash equilibrium (X, Y) of General Lotto game $\Gamma(a, b, u)$ where u is a bounded distance function with threshold u^T. If $\hat{u} = (4bu^T + 4b + u^T)(2u^T + 2)$, then we have $Pr(Y > \hat{u} + u^T) = 0$ and $Pr(X > \hat{u}) = 0$.*

Proof. First, we prove for any integer $z > \hat{u}$, $\Pr(X = z) = 0$. The proof is by contradiction. Let $z > \hat{u}$ be an integer with non-zero probability in X. Thus there is an integer $x < a$ with non-zero probability in X. Consider the pair distribution $T^a_{x,z}$. We define another pair distribution $T^a_{x,y}$ where $y = 4bu^T + 4b + u^T$.

Consider strategy $X^\epsilon = X - \epsilon T^a_{x,z} + \epsilon T^a_{x,y}$. Note that (X, Y) is a Nash equilibrium of the game. This means strategy X is a best response of player A to strategy Y of player B which implies $h^A_\Gamma(X, Y) \geq h^A_\Gamma(X^\epsilon, Y)$. On the other hand, because of the linearity of expectation we can write

$$h^A_\Gamma(X^\epsilon, Y) = h^A_\Gamma(X, Y) - \epsilon h^A_\Gamma(T^a_{x,z}, Y) + \epsilon h^A_\Gamma(T^a_{x,y}, Y).$$

Therefore, we conclude $w = h^A_\Gamma(T^a_{x,z}, Y) - h^A_\Gamma(T^a_{x,y}, Y) \geq 0$. Let $p = \alpha^a_{z,x}$ and $q = \alpha^a_{y,x}$. We have

$$w = \sum_{i=0}^{+\infty} \Pr(Y = i)[(1-p)u(x, i) + pu(z, i)] - \sum_{i=0}^{+\infty} \Pr(Y = i)[(1-q)u(x, i) + qu(y, i)] \geq 0.$$

We write w as $w = w_1 + w_2 - w_3 - w_4 + w_5$, where

$$
\begin{aligned}
w_1 &= \sum_{i=0}^{x} \Pr(Y = i)[[(1-p)u(x,i) + pu(z,i)] - [(1-q)u(x,i) + qu(y,i)]], \\
w_2 &= \sum_{i=x+1}^{+\infty} \Pr(Y = i)[(1-p)u(x,i) - (1-q)u(x,i)], \\
w_3 &= \sum_{i=x+1}^{y-u^T-1} \Pr(Y = i)qu(y,i), \\
w_4 &= \sum_{i=y-u^T}^{+\infty} \Pr(Y = i)qu(y,i), \\
w_5 &= \sum_{i=x+1}^{+\infty} \Pr(Y = i)pu(z,i).
\end{aligned}
$$

Since $1-p \geq 1-q$ and $u(z,i) = u(y,i) = u^M$ for all $i \leq x$, we can conclude $w_1 \leq 0$. For all $i > x$, we have $u(x,i) \leq 0$, and we also know $1-p \geq 1-q$. These mean $w_2 \leq 0$. Since for all $i \leq y - u^T$ we have $u(y,i) = u^M$, we conclude

$$w_3 = qu^M \sum_{i=x+1}^{y-u^T-1} \Pr(Y = i) \tag{2.19}$$

Moreover, for any arbitrary integers i and j, we have $-u^M \leq u(i,j) \leq u^M$. Thus

$$-w_4 \leq qu^M \sum_{i=y-u^T}^{+\infty} \Pr(Y = i) \tag{2.20}$$

$$w_5 \leq pu^M \sum_{i=x+1}^{+\infty} \Pr(Y = i) \tag{2.21}$$

Therefore by knowing $w \geq 0$, $w_1 \leq 0$, $w_2 \leq 0$, and considering Inequalities 2.19,

2.20, and 2.21, we conclude

$$u^M(-q\sum_{i=x+1}^{y-u^T-1}\Pr(Y=i)+q\sum_{i=y-u^T}^{+\infty}\Pr(Y=i)+p\sum_{i=x+1}^{+\infty}\Pr(Y=i))\geq w\geq 0. \tag{2.22}$$

$y-u^T-1=4bu^T+4b-1$ and by Lemma 23, $\sum_{i=0}^{y-u^T-1}\Pr(Y=i)\geq 1-\frac{b}{4bu^T+4b}=\frac{4u^T+3}{4u^T+4}$. Note that $x<a$ which means $\sum_{i=0}^{x}\Pr(Y=i)\leq\sum_{i=0}^{a-1}\Pr(Y=i)$. On the other hand, Lemma 24 says $\sum_{i=0}^{a-1}\Pr(Y=i)\leq\frac{u^T}{u^T+1}$. Hence we can conclude $\sum_{i=0}^{x}\Pr(Y=i)\leq\frac{u^T}{u^T+1}$. Therefore

$$\sum_{i=x+1}^{y-u^T-1}\Pr(Y=i)=\sum_{i=0}^{y-u^T-1}\Pr(Y=i)-\sum_{i=0}^{x}\Pr(Y=i)\geq\frac{3}{4u^T+4} \tag{2.23}$$

and

$$\sum_{i=y-u^T}^{+\infty}\Pr(Y=i)=1-\sum_{i=0}^{y-u^T-1}\Pr(Y=i)\leq\frac{1}{4u^T+4}. \tag{2.24}$$

By Inequalities 2.22, 2.23, 2.24, and $\sum_{i=x+1}^{+\infty}\Pr(Y=i)\leq 1$, we have

$$-q\frac{3}{4u^T+4}+q\frac{1}{4u^T+4}+p\geq 0. \tag{2.25}$$

which implies $\frac{q}{p}\leq 2u^T+2$. Recalling $p=\alpha^a_{z,x}=\frac{a-x}{z-x}$, $q=\alpha^a_{y,x}=\frac{a-x}{y-x}$, and $z>y$, we can bound $\frac{z}{y}$ as follows $\frac{z}{y}\leq\frac{z-x}{y-x}=\frac{q}{p}\leq 2u^T+2$. Therefore $z\leq y(2u^T+2)=\hat{u}$ which is a contradiction. Knowing that player A put zero probability on every number $z>\hat{u}$ and considering the definition of bounded distance function u, player B will put zero probability of every number greater than $\hat{u}+u^T$ in any Nash equilibrium. □

The following theorem follows immediately after Theorem 19 and Lemma 25.

Theorem 26. *There is a polynomial time algorithm which finds a Nash Equilibrium of the General Lotto game* $\Gamma(a, b, u)$ *where* u *is a bounded distance function.*

Proof. Let $\bar{u} = (4bu^T + 4b + u^T)(2u^T + 2) + u^T$. Lemma 25 shows there is a bound on the optimal strategies in a Nash equilibrium. More precisely $Pr(Y > \bar{u}) = 0$, where Y is a strategy of player A or B. Thus General Lotto game $\Gamma(a, b, u)$ is equivalent to finite General Lotto game $\Gamma(a, b, f, S)$, where $S = \{1, 2, \ldots, \bar{u}\}$. By Theorem 19, a polynomial-time algorithm finds a Nash equilibrium of the game. □

2.4 Oracles

In this section we describe, in precise detail, the separating oracles used by the ellipsoid method to solve our represented linear programs. Consider we are given a sequence $c_0, c_1, \ldots, c_{k(m+1)}$, where k is the number of battlefields and m is the number of troops for a player. We first present an algorithm which finds a pure strategy $x = (x_1, x_2, \ldots, x_k) \in \mathcal{X}$ such that $\sum_{i=1}^{k} x_i = m$, and $\hat{\mathbf{x}} = \mathcal{G}(x)$ minimizes the following equation.

$$c_0 + \sum_{i=1}^{k(m+1)} c_i \hat{x}_i \tag{2.26}$$

Then we leverage this algorithm and design polynomial-time algorithms for the hyperplane separating oracle and best-response separating oracle. The following lemma shows that Algorithm 1 (FindBestPure) finds the minimizer of Equation 2.26.

Algorithm 1: FindBestPure

input: m, k, $c_0, c_1, c_2, \ldots, c_{k(m+1)}$

for $j \leftarrow 1$ to m **do**
    $d[0, j] \leftarrow c_0$
end for
for $i \leftarrow 1$ to k **do**
    for $t \leftarrow 0$ to m **do**
        for $j \leftarrow 0$ to t **do**
            if $d[i-1, t-j] + c_{(i-1)(m+1)+j+1} < d[i, t]$ **then**
                $d[i, t] \leftarrow d[i-1, t-j] + c_{(i-1)(m+1)+j+1}$
                $r[i, t] \leftarrow j$
            end if
        end for
    end for
end for
$rem \leftarrow m$
for $i \leftarrow k$ downto 1 **do**
    $x_i \leftarrow r[i, rem]$
    $rem \leftarrow rem - r[i, rem]$
end for
return $x = (x_1, x_2, \ldots, x_k)$

Lemma 27. *Given two integers m and k and a sequence $c_0, c_1, \ldots, c_{k(m+1)}$, algorithm* FindBestPure *correctly finds an optimal pure strategy $x = (x_1, x_2, \ldots, x_k)$ where $\sum_{i=1}^{k} x_i = m$, $\hat{\mathbf{x}} = \mathcal{G}(x)$ and $\hat{\mathbf{x}}$ minimizes $c_0 + \sum_{i=1}^{k(m+1)} c_i \hat{x}_i$.*

Proof. In Algorithm FindBestPure , using a dynamic programming approach, we define $d[i,t]$ to be the minimum possible value of $c_0 + \sum_{i'=1}^{i(t+1)} c_{i'} \hat{x}_{i'}$ where $\sum_{i'=1}^{i} x_{i'} = t$. Hence, $d[k,m]$ denotes the minimum possible value of $c_0 + \sum_{i=1}^{k(m+1)} c_i \hat{x}_i$. Now, we show that Algorithm FindBestPure correctly computes $d[i,t]$ for all $0 \leq i \leq k$ and $0 \leq t \leq m$. Obviously $d[0,j]$ is equal to c_0. For an arbitrary $i > 0$ and t, the optimal strategy x puts $0 \leq t' \leq t$ units in the i-th battlefield and the applied cost in the equation 2.26 is equal to $c_{(i-1)(m+1)+t'+1}$. Thus,

$$d[i,t] = \min_{0 \leq t' \leq t} \{d[i-1, t-t'] + c_{(i-1)(m+1)+t'+1}\}$$

To compute the optimal pure strategy $x = (x_1, x_2, \ldots, x_k)$ we also keep a value

$$r[i,t] = \operatorname*{argmin}_{0 \leq t' \leq t} \{d[i-1, t-t'] + c_{(i-1)(m+1)+t'+1}\}$$

which determines the number of units the optimal strategy should put in the i-th battlefield to minimize $c_0 + \sum_{i'=1}^{i(t+1)} c_{i'} \hat{x}_{i'}$. Assuming we have correctly computed $x_{i+1}, \ldots, x_k$, in line 16, algorithm FindBestPure correctly computes x_i which is equal to $r[i, m - \sum_{j=i+1}^{k} x_j]$. Since $x_k = r[k,m]$ we can conclude algorithm FindBestPure correctly computes the optimal strategy $x = (x_1, x_2, \ldots, x_k)$. □

2.4.1 Hyperplane Separating Oracle

Algorithm 2 (HyperplaneOracle) gets a hyperplane as input and either finds a point in I_A which violates constraints in LP 2.5 or reports that all points in I_A are satisfying all constraints in LP 2.5. We suppose that the input hyperplane is described by the following equation,

$$\alpha_0 + \alpha_1 \hat{x}_1 + \ldots + \alpha_{k(a+1)} \hat{x}_{k(a+1)} = 0 \tag{2.27}$$

and we want to find a point $\hat{\mathbf{x}} \in I_A$ that violates the following constraint:

$$\alpha_0 + \sum_{i=1}^{n} \alpha_i \hat{x}_i \geq 0 \tag{2.28}$$

This problem is equivalent to finding a point $\hat{\mathbf{x}}^{min} \in I_A$ which minimizes equation $\alpha_0 + \sum_{i=1}^{n} \alpha_i \hat{x}_i$. If $\alpha_0 + \sum_{i=1}^{n} \alpha_i \hat{x}_i^{min} \geq 0$ it means all points in I_A are satisfying the constraints of LP, and otherwise $\hat{\mathbf{x}}^{min}$ is a point which violates constraint 2.6. Since points in I_A are equivalent to pure strategies of player A, we can use algorithm FindBestPure to find $\hat{\mathbf{x}}^{min}$. Thus we can conclude algorithm HyperplaneOracle correctly finds a violated constraint or reports that the hyperplane satisfies constraints 2.6 of LP 2.5.

Algorithm 2: HyperplaneOracle

input: a, k, $\alpha_0, \alpha_1, \ldots, \alpha_{k(a+1)}$

1: $x^{min} \leftarrow \textsc{FindBestPure}(a, k, \alpha_0, \alpha_1, \ldots, \alpha_{k(a+1)})$
2: $\hat{\mathbf{x}}^{min} \leftarrow \mathcal{G}_A(x^{min})$
3: **if** $\alpha_0 + \sum_{i=1}^{k(a+1)} \alpha_i \hat{x}_i^{min} \geq 0$ **then**
4: **return** pass
5: **else**
6: **return** $\alpha_0 + \sum_{i=1}^{k(a+1)} \alpha_i \hat{x}_i^{min} < 0$
7: **end if**

2.4.2 Best-response Separating Oracle

Algorithm 3 (BestRespOracle) gets a pair $(\hat{\mathbf{x}}, U)$ as input and decides whether there is a pure strategy $y = (y_1, y_2, \ldots, y_k) \in \mathcal{Y}$ such that for $\hat{\mathbf{y}} = \mathcal{G}_B(y)$, we have

$$\sum_{i=1}^{k} \sum_{t_a=0}^{a} \sum_{t_b=0}^{b} \hat{x}_{i,t_a} \hat{y}_{i,t_b} u_i^A(t_a, t_b) < U. \tag{2.29}$$

We can rewrite inequality 2.29 as follows:

$$\sum_{i=1}^{k} \sum_{t_b=0}^{b} \hat{y}_{i,t_b} \sum_{t_a=0}^{a} \hat{x}_{i,t_a} u_i^A(t_a, t_b) < U$$

Therefore, by letting $c_{i,t_b} = \sum_{t_a=0}^{a} \hat{x}_{i,t_a} u_i^A(t_a, t_b)$, this problem is equivalent to find a point $\hat{\mathbf{y}}^{min} \in I_B$ which minimizes $\sum_{i'=1}^{k(b+1)} c_{i'} \hat{y}_{i'}$. If $\sum_{i'=1}^{k(b+1)} c_{i'} \hat{y}_{i'}^{min} < U$ we have found a violating payoff constraint of LP 2.2 and if $\sum_{i'=1}^{k(b+1)} c_{i'} \hat{y}_{i'}^{min} \geq U$, pair $(\hat{\mathbf{x}}, U)$ satisfies all the payoff constraints of LP 2.2. Thus, by Lemma 27 we conclude that algorithm BestRespOracle correctly finds a violating payoff constraint of LP 2.2 or reports that $(\hat{\mathbf{x}}, U)$ satisfies all the payoff constrains.

Algorithm 3: BestRespOracle

input: a, b, k, U, $\hat{x}_1, \ldots, \hat{x}_{k(a+1)}$

for $i \leftarrow 1$ to k **do**
  for $t_b \leftarrow 0$ to b **do**
    $c_{(i-1)(b+1)+t_b+1} = c_{i,t_b} = \sum_{t_a=0}^{a} \hat{x}_{i,t_a}\ u_i^A(t_a, t_b)$
  end for
end for
$y^{min} \leftarrow \text{FindBestPure}(b, k, c_0, c_1, c_2, \ldots, c_{k(b+1)})$
$\hat{\mathbf{y}}^{min} \leftarrow \mathcal{G}_B(y^{min})$
if $\sum_{i=1}^{k(b+1)} c_i \hat{y}_i^{min} \geq U$ **then**
  return pass
else
  return $\sum_{i=1}^{k(b+1)} c_i \hat{y}_i^{min} < U$
end if

Chapter 3: A Faster Algorithm: Replacing Ellipsoid by Flows

Throughout this chapter we assume the number of battlefields is denoted by k and the number of troops of players A and B are denoted by a and b respectively. Also in some cases we use $\mathsf{cnt}_{\mathsf{troops}}$ to denote the number of troops of an unknown player.

Generally mixed strategies are shown by a probability vector over pure strategies. However at some points in this chapter we project this representation to another space that specifies probabilities to each battlefield and troop count pair. More precisely, we map a mixed strategy x of player A to $\mathcal{G}^A(x) = \hat{x} \in [0,1]^{d(A)}$ where $d(A) = k \times (a+1)$. We may abuse this notation for convenience and use $\hat{x}_{i,j}$ to show the probability the mixed strategy x puts j troops in the i-th battlefield. Note that this mapping is not one-to-one. Similarly, we define $\mathcal{G}^B(x)$ to map a mixed strategy x of player B to a point in $[0,1]^{d(B)}$ where $d(B) = k \times (b+1)$. Let $\mathcal{R}^A$ and $\mathcal{R}^B$ denote the set of all possible mixed strategies of A and B in a Nash equilibrium. We define $\mathcal{P}_A = \{\hat{x} \mid \exists x \in \mathcal{R}^A, \mathcal{G}^A(x) = \hat{x}\}$ and $\mathcal{P}_B = \{\hat{x} \mid \exists x \in \mathcal{R}^B, \mathcal{G}^B(x) = \hat{x}\}$ to be the set of all Nash equilibrium strategies in the new space for A and B respectively.

Multi-Resource Colonel Blotto is a generalization of Colonel Blotto where each player may have different types of resources. In MRCB, there are k battlefields and c

resource types. Players simultaneously distribute all their resources of all types over the battlefields. Let a_i and b_i denote the number of resources of type i player A and B respectively have. A pure strategy of a player would be a partition of his resources over battlefields. In other words, let $x_{i,j}$ and $y_{i,j}$ denote the amount of resources of type j, players A and B put in battlefield i respectively. A vector $x = \langle x_{1,1}, \ldots, x_{k,c} \rangle$ is a pure strategy of player A if for any $1 \leq j \leq c$, $\sum_{i=1}^{k} x_{i,j} = a_j$. Similarly a vector $y = \langle y_{1,1}, \ldots, y_{k,c} \rangle$ is a pure strategy of player B if for any $1 \leq j \leq c$, $\sum_{i=1}^{k} y_{i,j} = b_j$. Let $U^A(x, y)$ and $U^B(x, y)$ denote the payoff of A and B and let $U_i^A(x, y)$ and $U_i^B(x, y)$ show their payoff over the i-th battlefield respectively. Note that

$$U^A(x, y) = \sum_{i=1}^{k} U_i^A(x, y)$$

and

$$U^B(x, y) = \sum_{i=1}^{k} U_i^B(x, y).$$

On the other hand since MRCB is a zero-sum game $U_i^A(x, y) = -U_i^B(x, y)$. Similar to Colonel Blotto we define $\mathcal{R}_{\mathcal{M}}^A$ and $\mathcal{R}_{\mathcal{M}}^B$ to denote the set of all possible mixed strategies of A and B in a Nash equilibrium of MRCB and for any mixed strategy x for player A we define the mapping $\mathcal{G}_{\mathcal{M}}^A(x) = \hat{x} \in [0, 1]^{d^{\mathcal{M}}(A)}$ where $d^{\mathcal{M}}(A) = k \times (a_1 + 1) \ldots \times (a_c + 1p)$ and by $\hat{x}_{i,j_1,\ldots,j_c}$ we mean the probability that in mixed strategy x, A puts j_t amount of resource type t in the i-th battlefield for any t where $1 \leq t \leq c$. We also define the same mapping for player B, $\mathcal{G}_{\mathcal{M}}^B(x) = \hat{x} \in [0, 1]^{d^{\mathcal{M}}(B)}$ where $d^{\mathcal{M}}(B) = k \times (b_1 + 1) \ldots \times (b_c + 1)$. Lastly we define $\mathcal{P}_A^{\mathcal{M}} = \{\hat{x} \mid \exists x \in \mathcal{R}_{\mathcal{M}}^A, \mathcal{G}_{\mathcal{M}}^A(x) = \hat{x}\}$ and $\mathcal{P}_B^{\mathcal{M}} = \{\hat{x} \mid \exists x \in \mathcal{R}_{\mathcal{M}}^B, \mathcal{G}_{\mathcal{M}}^B(x) = \hat{x}\}$ to be the set of all Nash

equilibrium strategies after the mapping.

3.1 LP Formulation

In this section we explain the LP formulation of Colonel Blotto proposed in Chapter 2 and show how it can be reformulated in a more efficient way. Recall that in the Colonel Blotto game, we have two players A and B, each in charge of a number of troops, namely a and b respectively. Moreover, the game is played on k battlefields and every player's pure strategy is an allocation of his troops to the battlefields. Therefore, the number of pure strategies of the players is $\binom{a+k-1}{k-1}$ for player A and $\binom{b+k-1}{k-1}$ for player B.

The conventional approach to formulate the mixed strategies of a game is to represent every strategy by a vector of probabilities over the pure strategies. More precisely, a mixed strategy of a player is denoted by a vector of size equal to the number of his pure strategies, whose every element indicates the likelihood of taking a specific action in the game. The only constraint that this vector adheres to, is that the probabilities are non-negative and add up to 1. Such a formulation for Colonel Blotto requires a huge amount of space and computation, since the number of pure strategies of each player in this game is exponentially large.

To overcome this hardness, we propose a more concise representation that doesn't suffer from the above problem. This is of course made possible by taking a significant hit on the simplicity of the description. They suggest, instead of indicating the probability of taking every action in the representation, we only keep

track of the probabilities that a mixed strategy allocates a certain amount of troops to every battlefield. In other words, in the new representation, for every number of troops and any battlefield we have a real number, that denotes the probability of allocating that amount of troops to the battlefield. As a result, the length of the representation reduces from the number of pure strategies to $(a+1)k$ for player A and $(b+1)k$ for player B. This is indeed followed by a key observation: given the corresponding representations of the strategies of both players, one can determine the outcome of the game regardless of the actual strategies. In other words, the information stored in the representations of the strategies suffices to determine the outcome of the game.

In contrast to the conventional formulation, the exponential representation is much more complicated and not well-understood. For example, in order to see if a representation corresponds to an actual strategy in the conventional formulation, we only need to verify that all of the probabilities are non-negative and their total sum is equal to 1. exponential representation, however, is not trivial to verify. Apart from the trivial constraints such as the probabilities add up to 1 or the number of allocated troops matches the number of the player's troops, there are many other constraints to be met. Moreover, it is not even proven whether such a representation can be verified with a polynomial number of linear constraints.

We leverage the new representation to determine the equilibria of Colonel Blotto in polynomial time. Recall that in zero-sum games such as Colonel Blotto, the minmax strategies are the same as the maxmin strategies, and the game is in Nash Equilibrium if and only if both players play a maxmin strategy [64]. Roughly

speaking, the high-level idea is to find a mixed strategy which performs the best against every strategy of the opponent. By the equivalence of the minmax and maxmin strategies then, one can show such a strategy is optimal for that player. Therefore, the naive formulation of the equilibria of Blotto is as follows:

$$\begin{aligned} \max \quad & u \\ \text{s.t.} \quad & \hat{x} \text{ is a valid strategy for player A} \\ & U^A(\hat{x}, \hat{y}) \geq u \qquad \forall \hat{y} \end{aligned} \tag{3.1}$$

Note that, $\hat{x}$ is a vector of size $(a+1)k$ that represents a strategy of player A. Similarly, for every mixed strategy of player B, represented by $\hat{y}$, we have a constraint to ensure $\hat{x}$ achieves a payoff of at least u against $\hat{y}$. Notice that the only variables of the program are the probabilities encoded in vector $\hat{x}$. All other parameters are given as input, and hence appear as constant coefficients in the program. As declared, there are two types of constraints in Program 3.1. The first set of constraints ensures the validity of $\hat{x}$, and the second set of constraints makes sure $\hat{x}$ performs well against every strategy of player B. We call the first set *the membership constraints* and the second set *the payoff constraints*. Since for every mixed strategy, there exists a best response of the opponent which is pure, one can narrow dawn the payoff constraints to the pure strategies of player B.

The last observation is to show both types of the constraints are convex in the sense that if two strategy profiles $\hat{x_1}$ and $\hat{x_2}$ meet either set of constraints, then

$\frac{\hat{x_1}+\hat{x_2}}{2}$ is also a feasible solution for that set. This implies that Program 3.1 is indeed a linear program that can be solved efficiently via the ellipsoid method. However, our algorithm is practically impossible to run, as its computational complexity is $O(\mathsf{cnt}_{\mathsf{troops}}{}^{12}k^4)$.

The reason our algorithm is so slow is that their LP has exponentially many constraints. Therefore, they need to run the ellipsoid algorithm run solve the program. In addition to this, their separation oracle is itself a linear program with exponentially many constraints which is again very time consuming to run. However, a careful analysis shows that these exponentially many constraints are all necessary and none of them are redundant. This implies that the space of the LP as described in Chapter 2 requires exponentially many constraints to formulate and hence we cannot hope for a better algorithm. A natural question that emerges, however, is whether we can change the space of the LP to solve it with a more efficient algorithm?

In this chapter we answer the above question in the affirmative. There has been persistent effort to find efficient formulations for many classic polytopes. As an example, *spanning trees* of a graph can be formulated via a linear program that has an exponential number of linear constraints. It is also not hard to show none of those constraints are redundant [65]. However, Martin [66] showed that the same polytope can be formulated with $O(n^3)$ linear constraints where n is the number of nodes of the graph. Other examples are *the permutahedron* [67], *the parity polytope* [68], and *the matching polytope* [46]. In these examples, a substantial decrease in the number of constraints of the linear formulation of a problem is made possible by

adding auxiliary variables to the program. Our work follows the same guideline to formulate the equilibria of Blotto with a small number of constraints.

In Section 3.2, we explain how to formulate the membership and payoff limitations with a small number of linear constraints. Finally in Section 3.3, we show that our formulation is near optimal. In other words, we show that any linear program that formulates the equilibria of Blotto, has to have as many linear constraints as the number of constraints in our formulation within a constant factor. We show this via *rectangle covering lower bound* proposed by Yannakakis [69]

3.2 Main Results

In this section we give a linear program to find a maxmin strategy for a player in an instance of Colonel Blotto with polynomially many constraints and variables. To do this, we describe the same representation for exponential LP in another dimension, to reduce the number of constraints. This gives us a much better running time, since they had to use ellipsoid method to find a solution for their LP in polynomial time, which makes their algorithm very slow and impractical. We define a *layered graph* for each player and show any mixed strategy of a player can be mapped to a particular flow in his layered graph. Our LP includes two set of constraints, *membership constraints* and *payoff constraints*. Membership constraints guarantee we find a valid strategy and payoff constraints guarantee this strategy minimizes the maximum benefit of the other player.

Definition 28 (Layered Graph)**.** *For an instance of a Blotto game with k battle-*

fields, we define a layered graph for a player with $\mathsf{cnt}_{\mathsf{troops}}$ *troops as follows: The layered graph has* $k+1$ *layers and* $\mathsf{cnt}_{\mathsf{troops}}+1$ *vertices in each layer. Let* $v_{i,j}$ *denote the* j*'th vertex in the* i*'th layer (* $0 \leq i \leq k$ *and* $0 \leq j \leq \mathsf{cnt}_{\mathsf{troops}}$*). For any* $1 \leq i \leq k$ *there exists a directed edge from* $v_{i-1,j}$ *to* $v_{i,l}$ *iff* $0 \leq j \leq l \leq \mathsf{cnt}_{\mathsf{troops}}$*. We denote the layered graph of player* A *and* B *by* $\mathcal{L}^A$ *and* $\mathcal{L}^B$ *respectively.*

Based on the definition of layered graph we define *canonical paths* as follows:

Definition 29 (Canonical Path)**.** *A canonical path is a directed path in a layered graph that starts from* $v_{0,0}$ *and ends at* $v_{k,\mathsf{cnt}_{\mathsf{troops}}}$*.*

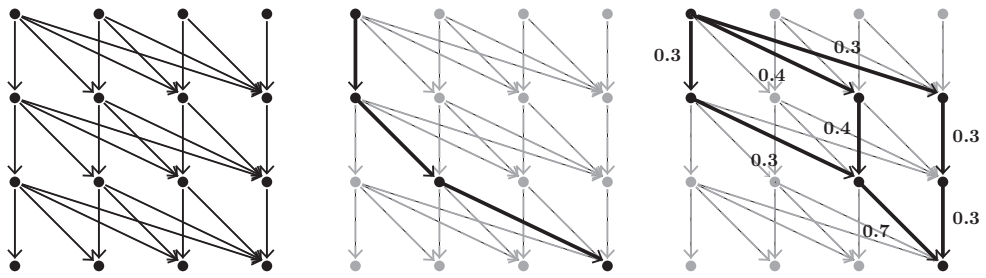

Figure 3.1: Figure (a) shows a layered graph for a player with 3 troops playing over 3 battlefields. In Figure (b) a canonical path corresponding to a pure strategy where the player puts no troops on the first battlefield, 1 troop on the second one and two troops on the 3rd one is shown. Figure (c) shows a flow of size 1, that is a representation of a mixed strategy consisting of three pure strategies with probabilities 0.3, 0.4 and 0.3.

Figure 3.1 shows a layered graph and a canonical path. Now, we give a one-to-one mapping between canonical paths and pure strategies.

Lemma 30. *Each pure strategy for a player is equivalent to exactly one canonical path in the layered graph of him and vice versa.*

Proof. Since the edges in the layered graph exist only between two consecutive layers, each canonical path contains exactly k edges. Let p be an arbitrary canonical path in the layered graph of a player with $\mathsf{cnt}_{\mathsf{troops}}$ troops. In the equivalent pure strategy put l_i troops in the battlefield i if p contains the edge between $v_{i-1,j}$ and $v_{i,j+l_i}$ for some j. By definition of the layered graph, we have $l_i \geq 0$. Also since p starts from $v_{0,0}$ and ends in $v_{k,\mathsf{cnt}_{\mathsf{troops}}}$ we have $\sum_{i=0}^{k} l_i = \mathsf{cnt}_{\mathsf{troops}}$. Therefore this strategy is a valid pure strategy.

On the other hand, let s be a valid pure strategy and let s_i denote the total number of troops in battlefields 1 to i in strategy s. We claim the set of edges between $v_{i-1,s_{i-1}}$ and v_{i,s_i} for $1 \leq i \leq k$ is a canonical path. Note that $s_0 = 0$ and $s_k = \mathsf{cnt}_{\mathsf{troops}}$ also the endpoint of any of such edges is the starting point of the edge chosen from the next layer, so we have constructed a valid canonical path. □

So far it is clear how layered graphs are related to pure strategies using canonical paths. Now we explain the relation between mixed strategies and flows of size 1 where $v_{0,0}$ is the source and $v_{k,\mathsf{cnt}_{\mathsf{troops}}}$ is the sink. One approach to formulate the mixed strategies of a game is to represent every strategy by a vector of probabilities over the pure strategies. Since based on Lemma 30 each pure strategy is equivalent to a canonical path in the layered graph; for any pure strategy s with probability $P(s)$ in a mixed strategy we assign a flow of size $P(s)$ to the corresponding canonical paths of s in the layered graph. All these paths begin and end in $v_{0,0}$ and $v_{k,\mathsf{cnt}_{\mathsf{troops}}}$ respectively. Therefore since $\sum P(s) = 1$ for all pure strategies of a mixed strategy, the size of the corresponding flow would be exactly 1.

Corollary 31. *For any mixed strategy of a player with* $\mathsf{cnt}_{\mathsf{troops}}$ *troops there is exactly one corresponding flow from vertex* $v_{0,0}$ *to* $v_{k,\mathsf{cnt}_{\mathsf{troops}}}$ *in the layered graph of that player.*

Note that although we map any given mixed strategy to a flow of size 1 in the layered graph, this is not a one-to-one mapping because several mixed strategies could be mapped to the same flow. However in the following lemma we show that this mapping is surjective.

Lemma 32. *For any flow of size* 1 *from* $v_{0,0}$ *to* $v_{k,\mathsf{cnt}_{\mathsf{troops}}}$ *in the layered graph of a player with* $\mathsf{cnt}_{\mathsf{troops}}$ *troops, there is at least one mixed strategy of that player with a polynomial size support that is mapped to this flow.*

Proof. First, note that we can decompose any given flow to polynomially many flow paths from source to sink [70]. A flow path is a flow over only one path from source to sink. One algorithm to find such decomposition finds a path p from source to sink in each step and subtracts the minimum passing flow through its edges from every edge in p. The steps are repeated until there is no flow from source to sink. Since the flow passing through at least one edge becomes 0 at each step, the total number of these paths will not exceed the total number of edges in the graph. This means the number of flow paths in the decomposition will be polynomial.

Now, given a flow of size 1 from $v_{0,0}$ to $v_{k,\mathsf{cnt}_{\mathsf{troops}}}$, we can basically decompose it to polynomially many flow paths using the aforementioned algorithm. The paths over which these flow paths are defined correspond to pure strategies and the amount of flow passing through each, corresponds to its probability in the mixed strategy.

□

Using the flow representation for mixed strategies and the shown properties for it, we give the first LP with polynomially many constraints and variables to find a maxmin strategy for any player in an instance of Colonel Blotto. Our LP consists of two set of constraints, the first set (membership constraints) ensures we have a valid flow of size 1. This means we will be able to map the solution to a valid mixed strategy. The second set of constraints are needed to ensure the minimum payoff of the player we are finding the maxmin strategy for, is at least u. Now, by maximizing u we will get a maxmin strategy. In the following theorem we prove $\mathcal{P}_A$ could be formulated with polynomially many constraints and variables. Note that one can swap a and b and use the same LP to formulate $\mathcal{P}_B$.

Theorem 33. *In an instance of Colonel Blotto, with k battlefields and at most $\mathsf{cnt}_{\mathsf{troops}}$ troops for each player, $\mathcal{P}_A$ could be formulated with $\Theta(\mathsf{cnt}_{\mathsf{troops}}{}^2 k)$ constraints and $\Theta(\mathsf{cnt}_{\mathsf{troops}}{}^2 k)$ variables.*

Proof. The high-level representation of our LP is as follows:

$$\begin{aligned} \max \quad & u \\ \text{s.t.} \quad & \hat{x} \text{ is a valid strategy for player A} \\ & U^B(\hat{x}, \hat{y}) \leq -u \qquad \forall \hat{y}. \end{aligned} \tag{3.2}$$

The strategies $\hat{x}$ and $\hat{y}$ are represented using a flow of size 1 in the layered graph of player A and B respectively. In Lemma 32 we proved any valid flow representation could be mapped to a mixed strategy.

1	0	0	0
0	1	0	0
0	0	1	0

(a)

0.3	0	0.4	0.3
0.7	0	0.3	0
0.3	0.7	0	0

(b)

Figure 3.2: Figure (a) shows $\Pr(k = i)$ for the pure strategy specified in Figure 3.1-b and Figure (b) shows $\Pr(k = i)$ for the mixed strategy specified in Figure 3.1-c. The rows correspond to battlefields and the columns correspond to the number of troops.

To ensure we a have a valid flow of size 1 from $v_{0,0}$ to $v_{k,a}$ in $\mathcal{L}^A$ (recall that $\mathcal{L}^A$ denotes the layered graph of player A), we use the classic LP representation of flow [71]. That is, not having any negative flow and the total incoming flow of each vertex must be equal to its total outgoing flow except for the source and the sink. We denote the amount of flow passing through the edge from $v_{k,i}$ to $v_{k+1,j}$ by variable $F_{k,i,j}$. The exact membership constraints are shown in Linear Program 3.1-a.

On the other hand, we maximize the guaranteed payoff of player A, by bounding the maximum possible payoff of player B. To do this, first note that for any given strategy of player A, there exists a pure strategy for player B, that maximizes his payoff. Let $\Pr(k = j)$ denote the probability that player A puts j troops in the k-th battlefield. Figure 3.2 shows the value of $\Pr(k = j)$ for the illustrated examples in Figure 3.1. We can compute these probabilities using the variables defined in the previous constraints, as follows:

$$\Pr(k = i) = \sum_{i=0}^{a-j} F_{k,i,i+j} \tag{3.3}$$

By having these probabilities we can compute the expected payoff that player B

gets over battlefield k, if he puts i troops in it. Moreover consider a given canonical path p in $\mathcal{L}^B$ and let s_p be the pure strategy of player B, equivalent to p. We use $W^B_{k,i}$ to denote the expected payoff of player B over battlefield k by putting i troops in it. This means the expected payoff of playing strategy s_p would be $\sum W^B_{k,j-i}$ for any k, i and j such that there exists an edge from $v_{k,i}$ to $v_{k+1,j}$ in p. It is possible to compute $W^B_{k,i}$ using the following equation:

$$W^B_{k,i} = \sum_{l=0}^{a} \Pr(k = l) \times U^B_k(i, l) \qquad \forall k : 1 \leq t \leq k \tag{3.4}$$

Note that both equations to compute Pr(k = i) and $W^B_{k,i}$ are linear and could be computed in our LP.

Assume $W^B_{k,i}$ is the weight of the edge from $v_{k,j}$ to $v_{k+1,i+j}$ in $\mathcal{L}^B$. Given the probability distribution of player A (which we denoted by Pr(k = i)), the problem of finding the pure strategy of B with the maximum possible expected payoff, would be equivalent to finding a path from $v_{0,0}$ to $v_{k,b}$ with the maximum weight.

To find the path with the maximum weight from $v_{0,0}$ to $v_{k,b}$, we define an LP variable $D^B_{k,i}$ where its value is equal to the weight of the maximum weighted path from $v_{0,0}$ to $v_{k,i}$ and we update it using a simple dynamic programming like constraint:

$$D^B_{k,i} \geq D^B_{k-1,j} + W^B_{k-1,i-j} \qquad \forall i, j : 0 \leq j \leq i \leq b$$

The maximum weighted path from $v_{0,0}$ to $v_{k,b}$ would be equal to the value of $D^B_{k,b}$. The detailed constraints are shown in Linear Program 3.1-b.

$$\max \quad u$$

$$\text{(a)} \quad \begin{cases} \Sigma_{i=0}^{l} F_{k,i,l} = \Sigma_{j=l}^{a} F_{k+1,l,j} & \forall k, l : 1 \leq k \leq k-1, 0 \leq l \leq a \\ F_{k,i,j} \geq 0 & \forall k, i, j : 1 \leq k \leq k, 0 \leq i \leq j \leq a \\ \Sigma_{j=l}^{a} F_{1,l,j} = 0 & \forall l : 0 < l \leq a \\ \Sigma_{j=0}^{a} F_{1,0,j} = 1 & \\ \Sigma_{j=0}^{a} F_{k,j,a} = 1 & \end{cases}$$

$$\text{(b)} \quad \begin{cases} \Pr(k = i) = \Sigma_{i=0}^{a-j} F_{k,i,i+j} & \forall k, j : 1 \leq k \leq k, 0 \leq j \leq a \\ W_{k,i}^{B} = \Sigma_{l=0}^{a} \Pr(k = l) \times U_{k}^{B}(i, l) & \forall k, i : 1 \leq k \leq k, 0 \leq i \leq b \\ D_{0,i}^{B} = 0 & \forall i : 0 \leq i \leq b \\ D_{k,i}^{B} \geq D_{k-1,j}^{B} + W_{k-1,i-j}^{B} & \forall i, j : 0 \leq j \leq i \leq b \\ D_{k,b}^{B} \leq -u & \end{cases}$$

Linear Program 3.1: The detailed linear program to find a maxmin strategy for player A. The first set of constraints denoted by (a) ensure we get a valid flow of size 1 from $v_{0,0}$ to $v_{k,a}$ in the layered graph of player A (a mixed strategy of him) and the second set of constraints denoted by (b) ensure the guaranteed payoff of player A is at least u. The value of variable $F_{k,i,j}$ is the amount of flow passing through the edge from $v_{k,i}$ to $v_{k+1,j}$ for any valid k, i and j. Variable $D_{i,j}^{B}$ is the size of the maximum weighted path from $v_{0,0}$ to $v_{i,j}$ in the layered graph of player B, therefore $D_{k,b}^{B}$ denotes the maximum payoff of B and u is the guaranteed payoff of player A. For an informal explanation of the LP see the text.

Note that the number of variables we use in Linear Program 3.1 is as follows:

- Variables of type $F_{k,i,l}$: $\Theta(a^2k)$.
- Variables of type Pr(k = i) : $\Theta(ak)$.
- Variables of type $W^B_{k,i}$: $\Theta(bk)$.
- Variables of type $D^B_{k,i}$: $\Theta(bk)$.

Therefore the total number of variables is $\Theta(\mathsf{cnt}_{\mathsf{troops}}{}^2k)$. Also note that the number of non-negativity constraints ($F_{k,i,j} \geq 0$) is more than any other constraints and is $\Theta(\mathsf{cnt}_{\mathsf{troops}}{}^2k)$, therefore the total number of constraints is also $\Theta(\mathsf{cnt}_{\mathsf{troops}}{}^2k)$. □

To obtain a mixed strategy for player A, it suffices to run Linear Program 3.1 and find a mixed strategy of A that is mapped to the flow it finds. Note that based on Lemma 32 such mixed strategy always exists. Afterwards we do the same for player B by simply substituting a and b in the LP.

3.3 Lower Bound

A classic approach to reduce the number of LP constraints needed to describe a polytope is to do it in a higher dimension. More precisely, adding extra variables might reduce the number of facets of a polytope. This means a complex polytope may be much simpler in a higher dimension. This is exactly what we did in Section 3.2 to improve the previous algorithm. In this section we prove that any LP formulation that describes solutions of a Blotto game requires at least $\Theta(\mathsf{cnt}_{\mathsf{troops}}{}^2k)$

constraints, no matter what the dimension is. This proves the given LP in Section 3.2 is tight up to constant factors.

The minimum needed number of constraints in any formulation of a polytope P is called *extension complexity* of P, denoted by $\text{xc}(P)$. It is not usually easy to prove a lower bound directly on the extension complexity, because all possible formulations of the polytope must be considered. A very useful technique given by Yannakakis [69] is to prove a lower bound on the *positive rank* of the *slack matrix* of P which is proven to be equal to $\text{xc}(P)$. Note that you could define the slack matrix over any formulation of P and its positive rank would be equal to $\text{xc}(P)$, which means you do not have to worry about all possible formulations. To prove this lower bound we use a method called *rectangle covering lower bound*, already given in Yannakakis's paper. We will now formally define some of the concepts we used:

Definition 34 (Extension Complexity)**.** *Extension complexity of a polytope P, denoted by $\text{xc}(P)$ is the smallest number of facets of any other higher dimensional polytope Q that has a linear projection function π with $\pi(Q) = P$.*

The next concept we need is slack matrix, which is a matrix of non-negative real values where its columns correspond to vertices of P and its rows correspond to its facets. The value of each element of slack matrix is basically the distance of the vertex corresponding to its column from the facet corresponding to its row. More formally:

Definition 35 (Slack Matrix)**.** *Let $\{v_1, \ldots, v_v\}$ be the set of vertices of P and let*

$\{x \in \mathbb{R}^n | Ax \leq b\}$ *be the description of it. The slack matrix of* P *denoted by* S^P*, is defined by* $S^P_{ij} = b_i - A_i v_j$.

Also, the non-negative rank of a matrix S is the minimum number m such that S could be factored into two non-negative matrices F and V with dimensions $f \times m$ and $m \times v$.

Definition 36 (Non-negative Rank)**.** *We define the non-negative rank of a matrix* S *with* f *rows and* v *columns, denoted by* $\mathrm{rk}_+(S)$ *to be:*

$$\mathrm{rk}_+(S) = \min\{m | \exists F \in \mathbb{R}^{f \times m}_{\geq 0}, V \in \mathbb{R}^{m \times v}_{\geq 0} : S = FV\} \tag{3.5}$$

Yannakakis [69] proved that $\mathrm{xc}(P) = \mathrm{rk}_+(S^P)$. Therefore instead of proving a lower bound on the extension complexity of P, it only suffices to prove a lower bound on the positive rank of the corresponding slack matrix. As mentioned before, to do so, we will use the rectangle covering lower bound. A rectangle covering for a given non-negative matrix S is the minimum number of rectangles needed, to cover all the positive elements of S and none of its zeros (Figure 3.3), formally defined as follows:

Definition 37 (Rectangle Covering)**.** *Suppose* $r = \mathrm{rk}_+(S)$ *and let* $S = UV$ *be a factorization of* S *by non-negative matrices* U *and* V*. Let* $\mathrm{supp}(S)$ *denote the set of all the positive values of* S*. Then*

$$\mathrm{supp}(S) = \bigcup_{l=1}^{r} (\{i : U_{il} > 0\} \times \{j : V_{lj} > 0\})$$

is a rectangle covering of S with r rectangles.

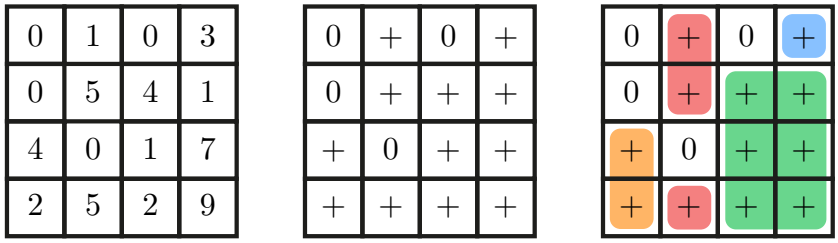

Figure 3.3: Figure (a) shows a sample matrix, in Figure (b) we change any non-negative value in the matrix of Figure (a) to "+" and in Figure (c) all these non-negative elements are covered by the minimum possible number of rectangles. Note that the non-negative rank of the matrix in Figure (a) could not be less than 4.

Yannakakis showed that the number of rectangles in a minimum rectangle covering could never be greater than $\mathrm{rk}_+(S)$, using a very simple proof. This means any lower bound of it, is also a lower bound of the actual $\mathrm{rk}_+(S)$. This is the technique we use in the proof of the following lemma, which is used later to prove the main theorem:

Lemma 38. *The extension complexity of the membership polytope of a player in an instance of Blotto with k battlefields and $\mathsf{cnt}_{\mathsf{troops}}$ troops for each player is at least $\Theta(\mathsf{cnt}_{\mathsf{troops}}{}^2 k)$.*

Proof. Assume w.l.g. that we are trying to describe the polytope of all valid strategies of player A, denoted by P. One way of describing this polytope was explained in the LP described in Section 3.2. Now from its membership constraints, only consider the ones that ensure the non-negativity of the flow passing through the edges

of the layered graph of player A:

$$F_{i,j,t} \geq 0 \qquad \forall i, j, t : 0 \leq i \leq k-1, 0 \leq j \leq j+t \leq a \tag{3.6}$$

From now on, only consider the part of the slack matrix corresponding to these constraints (we may occasionally call it the slack matrix), its columns as mentioned before, correspond to the vertices of the polytope, which in this case are all possible pure strategies of player A. Also its rows correspond to the mentioned constraints. Recall that any pure strategy is a canonical path in the layered graph of player A. Note that the slack matrix element corresponding to any arbitrarily chosen non-negativity constraint $e \geq 0$ and any arbitrary vertex v_j corresponding to a pure strategy S is 0 iff the equivalent canonical path of S does not contain e and is 1 if it does; since the elements of the slack matrix are calculated using the formula $S^P_{ij} = b - A_i v_j$ and in this case b is always zero and $A_i v_j$ is -1 iff S contains the edge in the constraint and is zero otherwise. This implies it is only consisted of zero and one values.

We call any edge $F_{b,i,j}$ with $j - i > \frac{\mathsf{cnt}_{\mathsf{troops}}}{2}$ a long edge. A canonical path may only contain at most one such edge. On the other hand, any rectangle in the rectangle covering is basically a set of vertices and a set of constraints. Note that all the equivalent pure strategies of those vertices must contain the edges over which the constraints are defined. Therefore no rectangle could contain more than one

constraint over long edges. The number of long edges in the layered graph is exactly

$$\frac{k(\mathsf{cnt}_{\mathsf{troops}} - \lceil \frac{\mathsf{cnt}_{\mathsf{troops}}+1}{2} \rceil)(\mathsf{cnt}_{\mathsf{troops}} - \lceil \frac{\mathsf{cnt}_{\mathsf{troops}}+1}{2} \rceil + 1)}{2}. \tag{3.7}$$

Therefore the minimum number of rectangles to cover all non-negative elements of the slack matrix is at least of the same size and therefore $\Theta(\mathsf{cnt}_{\mathsf{troops}}{}^2 k)$. □

Theorem 39. *In an instance of Blotto with k battlefields and $\mathsf{cnt}_{\mathsf{troops}}$ troops for each player the extension complexity of $\mathcal{P}_A$ is $\Theta(\mathsf{cnt}_{\mathsf{troops}}{}^2 k)$.*

Proof. Assume the utility function is defined as follows:

$$U^A(\hat{x}, \hat{y}) = 0 \qquad \forall \hat{x}, \hat{y}. \tag{3.8}$$

This means any possible strategy is a maxmin strategy for both players. In particular, the polytope of all possible maxmin strategies of any arbitrarily chosen player of this game, denoted by P contains all possible valid strategies. Now using Lemma 38 we know $\mathrm{xc}(P)$ is at least $\Theta(\mathsf{cnt}_{\mathsf{troops}}{}^2 k)$. On the other hand, in Section 3.2 we gave an LP with $\Theta(\mathsf{cnt}_{\mathsf{troops}}{}^2 k)$ constraints to formulate the maxmin polytope, therefore the extension complexity of it is exactly $\Theta(\mathsf{cnt}_{\mathsf{troops}}{}^2 k)$. □

3.4 Multi-Resource Colonel Blotto

In this section we explain how our results could be generalized to solve Multi-Resource Colonel Blotto, or *MRCB*. We define MRCB to be exactly the same game as Colonel Blotto, except instead of having only one type of resource (troops), players

may have any constant number of resource types. Examples of resource types would be time, money, energy, etc.

To solve MRCB we generalize some of the concepts we defined for Colonel Blotto. We first define generalized layered graphs and generalized canonical paths as follows:

Definition 40 (Generalized Layered Graph)**.** *Let* $\mathsf{cnt}_{\mathsf{troops}_m}$ *denote the total number of available resources of* m*-th resource type for player* X*. The generalized layered graph of* X *has* $k \times \mathsf{cnt}_{\mathsf{troops}1} \times \ldots \times \mathsf{cnt}_{\mathsf{troops}c}$ *vertices denoted by* $v(i, r_1, \ldots, r_c)$*, with a directed edge from* $v(i, r_1, \ldots, r_{m-1}, x, r_{m+1}, \ldots, r_c)$ *to* $v(i{+}1, r_1, \ldots, r_{m-1}, y, r_{m+1}, \ldots, r_c)$ *for any possible* i*,* r *and* $0 \leq x \leq y \leq \mathsf{cnt}_{\mathsf{troops}_m}$*.*

Definition 41 (Generalized Canonical Path)**.** *A generalized canonical path is defined over a generalized layered graph and is a directed path from* $v_{0,0,\ldots,0}$ *to* $v_{k,\mathsf{cnt}_{\mathsf{troops}_1},\ldots,\mathsf{cnt}_{\mathsf{troops}_c}}$*.*

By these generalization we can simply prove that pure strategies of a player are equivalent to canonical paths in his generalized layered graph and there could be a surjective mapping from his mixed strategies to flows of size 1 from $v(0, \ldots, 0)$ to $v(k, \mathsf{cnt}_{\mathsf{troops}_1}, \ldots, \mathsf{cnt}_{\mathsf{troops}_c})$ using similar techniques we used in Section 3.2.

Lemma 42. *Each pure strategy for a player in an instance of MRCB is equivalent to exactly one generalized canonical path in the generalized layered graph of him and vice versa.*

Lemma 43. *For any flow* f *of size* 1 *from* $v(0, \ldots, 0)$ *to* $v(k, \mathsf{cnt}_{\mathsf{troops}_1}, \ldots, \mathsf{cnt}_{\mathsf{troops}_c})$ *in the generalized layered graph of a player with* $\mathsf{cnt}_{\mathsf{troops}i}$ *troops of type* i*, there is at least one mixed strategy with a polynomial size support that is mapped to* f*.*

Using these properties, we can prove the following theorem:

Theorem 44. *In an instance of MRCB, $\mathcal{P}_A^{\mathcal{M}}$ could be formulated with $O(\mathsf{cnt_{troops}}^{2c}k)$ constraints and $\Theta(\mathsf{cnt_{troops}}^{2c}k)$ variables.*

Proof. The linear program would again look like this:

$$\begin{aligned} \max \quad & u \\ \text{s.t.} \quad & \hat{x} \text{ is a valid strategy for player A} \\ & U^B(\hat{x}, \hat{y}) \leq -u \qquad \forall \hat{y} \end{aligned} \tag{3.9}$$

For the first set of constraints (membership constraints) we can use the flow constraints over the generalized layered graph of player A to make sure we have a valid flow of size 1 from $v(0, \ldots, 0)$ to $v(K, \mathsf{cnt_{troops}}_1, \ldots, \mathsf{cnt_{troops}}_c)$. And for the second constraint (payoff constraint) we can find the maximum payoff of player B using a very similar set of constraints to the described one in Section 3.2, but over the generalized layered graph of player B. □

We can also prove the following lowerbound for MRCB.

Theorem 45. *In an instance of MRCB, the extension complexity of $\mathcal{P}_A^{\mathcal{M}}$ is $\Theta(\mathsf{cnt_{troops}}^{2c}k)$.*

Proof. The proof is very similar to the proof of Theorem 39. We only consider the rectangle covering lower bound over the part of the slack matrix corresponding to the non-negativity of flow through edges in the maxmin. We call an edge from $v(i, r_1, \ldots, r_{m-1}, x, r_{m+1}, \ldots, r_c)$ to $v(i{+}1, r_1, \ldots, r_{m-1}, y, r_{m+1}, \ldots, r_c)$ long if $y{-}x >$

$\frac{n_m}{2}$. No generalzied canonical path could contain more than c long edges therefore no rectangle could cover more than c constraints. On the other hand there are $\Theta(\mathsf{cnt}_{\mathsf{troops}}{}^{2c}k)$ long edges in the layered graph. Since c is a constant number the extension complexity is $\Omega(\mathsf{cnt}_{\mathsf{troops}}{}^{2c}k)$. Moreover since we already gave a possible formulation with $O(\mathsf{cnt}_{\mathsf{troops}}{}^{2c}k)$ constraints in Theorem 3.4 the extension complexity is also $O(\mathsf{cnt}_{\mathsf{troops}}{}^{2c}k)$ and therefore $\Theta(\mathsf{cnt}_{\mathsf{troops}}{}^{2c}k)$. □

3.5 Experimental Results

We implemented the algorithm described in Section 3.2 using Simplex method to solve the LP. We ran the code on a machine with a dual-core processor and an 8GB memory. The running time and the number of constraints of the LP for each input is shown in Table 3.1. Using this fast implementation we were able to run the code for different cases. In this section we will mention some of our observations that mostly confirm the theoretical predications.

An instance of Colonel Blotto is symmetric if the payoff function is the same for all battlefields, or in other words for any pure strategies x and y for player A and B and for any two battlefields i and j, $U_i^A(x,y) = U_j^A(x,y)$. Also, an instance of blotto is auctionary if the player allocating more troops in a battlefield wins it (gets more payoff over that battlefield). More formally in an auctionary instance of Colonel Blotto, if x and y are some pure strategies for player A and B respectively,

k	a	b	Constraints	Running Time
10	20	20	3595	0m3.575s
10	20	25	4855	0m3.993s
10	20	30	6365	0m6.695s
10	25	25	5295	0m8.245s
10	25	30	6805	0m7.502s
10	30	30	7320	0m30.955s
15	20	20	5065	0m14.965s
15	20	25	6950	0m11.842s
15	20	30	9210	0m24.196s
15	25	25	7440	0m46.165s
15	25	30	9700	0m31.714s
15	30	30	10265	2m20.776s
20	20	20	6535	0m46.282s
20	20	25	9045	0m35.758s
20	20	30	12055	0m38.507s
20	25	25	9585	1m38.367s
20	25	30	12595	0m51.795s
20	30	30	13210	9m13.288s

Table 3.1: The number of constraints and the running time of the implemented Colonel Blotto based on different inputs. The first column shows the number of battlefields, the second and third columns show the number of troops of player A and B respectively. The number of constraints does not include the non-negativity constraints since by default every variable was assumed to be non-negative in the library we used.

then

$$U_i^A(x,y) = \begin{cases} +w(i), & \text{if } x_i > y_i \\ 0, & \text{if } x_i = y_i \\ -w(i), & \text{otherwise} \end{cases}$$

Recall that x_i and y_i denote the amount of troops A and B put in the i-th battlefield respectively.

Note that in an auctionary Colonel Blotto if $a \geq (b+1)k$, then by putting $b+1$ troops in each battlefield, player A wins all the battlefields and gets the maximum possible overall payoff. On the other hand if $a = b$, the payoff of player A in any Nash equilibrium is exactly 0 because there is no difference between player A and player B by definition of an auctionary Colonel Blotto if $a = b$, and any strategy for A could also be used for B and vice versa. W.l.g. we can ignore the case where $a < b$. However, it is not easy to guess the payoff of A in a Nash equilibrium if $b \leq a < (b+1)k$. After running the code for different inputs, we noticed the growth of U^A with respect to a (when b is fixed) has a common shape for all inputs. Figure 3.4 shows the chart for different values of a, b and $\mathsf{cnt}_{\mathsf{troops}}$.

There has been several attempts to mathematically find the optimum payoff of players under different conditions. For example Roberson [34] considered the continuous version of Colonel Blotto and solved it. Hart [36] solved the symmetric and auctionary model and solved it for some special cases. Little is known about whether it is possible to completely solve the discrete version when the game is symmetric and auctionary or not.

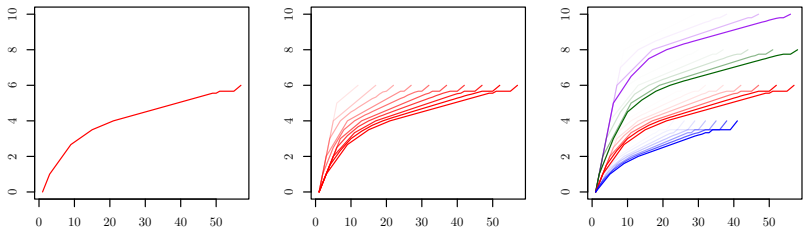

Figure 3.4: The y-axis is the payoff of A in the Nash equilibrium and the x-axis shows the value of $a - b$. In Figure (a), $k = 6$ and $b = 10$. In Figure (b), $k = 6$ and for different values of b in the range of 1 to 10 the same diagram as Figure (a-) is drawn. Figure (c) is the same plot as Figure (b) but for different values of k. For instance for the blue lines $k = 4$, for the red lines $k = 6$, for the green lines $k = 8$ and for the purple lines $k = 10$. In all examples payoff function of player A over a battlefield i, is $sgn(x_i - y_i)$ where x_i and y_i denote the number of troops A and B put in the i-th battlefield respectively.

Surprisingly, we observed the payoff of players in the symmetric and auctionary discrete version, is very close to the continuous version Roberson considered. The payoffs are specially very close when the number of troops are large compared to the number of battlefields, making the strategies more flexible and more similar to the continuous version. Figure 3.5 compares the payoffs in the aforementioned models. In Roberson's model in case of a tie, the player with more resources wins while in the normal case there is no such assumption; however a tie rarely happens since by adding any small amount of resources the player losing the battlefield would win it.

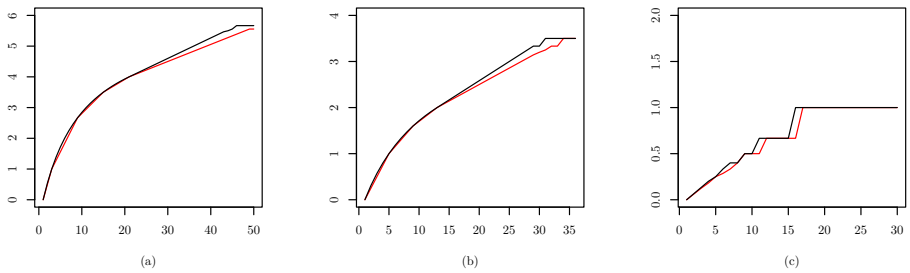

Figure 3.5: The y-axis is the payoff of A in the Nash equilibrium and the x-axis shows the value of $a - b$. The black and red line show the payoff in the continuous model and discrete model respectively. In figure (a), $k = 6$ and $b = 10$, in figure (b), $k = 4$ and $b = 12$ and in figure (c), $k = 2$ and $b = 30$.

Chapter 4: Probability Maximization Versus Expectation Maximization

Throughout this chapter, we consider the *discrete* and *continuous* variants of Colonel Blotto, as well as the auditing game and provide approximately optimal (u, p)-maxmin strategies for these games. Our main result is an algorithm with logarithmic approximation factor for the discrete Colonel Blotto game. Next, we provide a constant approximation algorithm for the *continuous* variant of Colonel Blotto and finally we provide a constant approximation algorithm for the auditing game.

At a high level, our techniques are inspired by recent developments in game theory and optimization. For instance, when the goal is to maximize the guaranteed payoff of a player (finding a $(u, 1)$-maxmin strategy) our problem settings generalizes Stackelberg games. In addition to this, when the goal is to find a (u, p)-maxmin strategy for arbitrary $0 \leq p \leq 1$, the problem extends the robust optimization problem studied in [72]. We also devise a decomposition technique inspired by [73–76].

We recall that a plethora of studies have analyzed and characterized the equilibria of structured zero-sum games such as Colonel Blotto [1,2,53,77–79]. In partic-

ular, we present in Chapter 2 a polynomial time algorithm to compute the maxmin strategies of Colonel Blotto. Provided that the maxmin strategies of Colonel Blotto are available, it is crucial to understand how well such strategies perform when the objective is *not* to maximize the expected payoff but to approximate a (u, p)-maxmin strategy. We begin in Section 4.2, by illustrating the difference between the maxmin strategies and (u, p)-maxmin strategies. Although we show that in special cases, a maxmin strategy provides a decent approximation of a (u, p)-maxmin strategy, we present an example to show that in general, maxmin strategies are not competitive to the (u, p)-maxmin ones. Our counter-example is a Colonel Blotto game in which the number of troops of player B is many times more than the troops of player A.

Theorem 46 [restated]. *For any given u, p and arbitrarily small constants $0 < \alpha < 1$ and $0 < \beta < 1$, there exists an instance of Colonel Blotto (both discrete and continuous), where for any approximate $(\alpha' u, \beta' p)$-maxmin strategy that an expectation maximizer algorithm returns, either $\alpha' < \alpha$ or $\beta' < \beta$.*

Theorem 46 states that in order to provide an exact or even an approximation algorithm for (u, p)-maxmin strategies, one needs to go beyond the expectation maximizer algorithms. Following this observation, we begin our results by studying the special case of $(u, 1)$-maxmin or in other words the *u-guaranteed payoff* strategies for the discrete variant of Colonel Blotto game.

For the special case of $p = 1$, one possible strategy of the opponent is to randomize over all pure strategies. Thus, any strategy of player A that guarantees

a payoff of at least u with probability one, must obtain a payoff of at least u against any pure strategy of the opponent. Indeed this condition is sufficient to declare a strategy (u, p)-maxmin or in other words, any strategy of player A that obtains a payoff of at least u against any pure strategy of player B is $(u, 1)$-maxmin. Moreover, one can show that randomization offers no benefit to player A when the objective is to find a $(u, 1)$-maxmin strategy. Therefore, the definition of $(u, 1)$-maxmin strategies coincides with the notion of *pure maxmin* strategies. Based on this, our objective is to find a pure strategy for player A that obtains the maximum payoff against any best response of the opponent. This is very similar to Stackelberg games with the exception that here we only incorporate the pure strategies of player A.

For a fixed strategy of player A, the best response of player B can be modeled as a knapsack problem. Let a_i denote the number of troops of player A in battlefield i. In order for player B to maximize her payoff (or equivalently minimize player A's payoff) she needs to find a subset of battlefields S and put a_i troops in every battlefield i in this subset. The constraint is that she can only afford to put M troops in those battlefields and therefore $\sum_{i \in S} a_i$ should be bounded by M. Therefore the problem is to find a subset S of battlefields with the maximum total weight subject to $\sum_{i \in S} a_i \leq M$. This problem can be solved in time $\mathsf{poly}(N, M, K)$ with a classic knapsack algorithm. However, a polytime algorithm for best response does not lead to a polytime solution since player A has exponentially many pure strategies and verifying all such strategies takes exponential time.

To overcome this challenge, we relax the best response algorithm of player B to an almost best response greedy algorithm. Let $\mathsf{W}_{\mathsf{max}} = \max w_i$ be the maximum

weight of a battlefield or equivalently the maximum profit of an item in the knapsack problem. It is well-known that the following greedy algorithm for knapsack guarantees an additive error of at most $\mathsf{W}_{\mathsf{max}}$ in comparison to the optimal solution: sort the items based on the ratio of profit over size and put these items into the knapsack accordingly. Based on this observation, if we restrict the opponent to play according to the greedy algorithm, the performance of our solution drops by an additive factor of at most $\mathsf{W}_{\mathsf{max}}$. Once we replace the strategy of player B by the greedy knapsack algorithm, finding a maxmin strategy of player A becomes tractable. More precisely, we show that the problem of finding an optimal strategy for player A against the greedy knapsack algorithm boils down to a dynamic program that can be solved in polynomial time.

In order to turn the $\mathsf{W}_{\mathsf{max}}$ additive error into a $1/2$ multiplicative error, we also consider a strategy of player A that puts all her troops in the battlefield with the highest weight. We show that the better of the two strategies guarantees a profit of at least $u/2$ against any strategy of the opponent where u is the maximum possible guaranteed payoff of player A.

Theorem 50 [restated]. There exists a polynomial time algorithm that gives a $(u/2, 1)$-maxmin strategy of player A, assuming that u is the maximum guaranteed payoff of player A.

In Section 4.3.2, we consider the problem of approximating a (u, p)-maxmin strategy for an arbitrary u and $0 \leq p \leq 1$. This case is more challenging than the case

of guaranteed payoff since (1) the solution is not necessarily a pure strategy; and (2) the knapsack modeling for the best response of player B is no longer available. We begin by considering the special case of uniform weights (wherein all the weights are equal to 1) and providing an algorithm for approximating a (u, p)-maxmin strategy in this setting. Later, we reduce the case of $0 \leq p \leq 1$ to this case. Since all the weights are equal to 1, we are able to characterize the optimal strategies of the players and based on that we provide simple strategies that obtain a fraction of the guarantees that the optimal strategies provide.

Lemma 57 [restated]. *Given that there exists a (u, p)-maxmin strategy for player A in an instance of discrete Colonel Blotto with uniform weights, there exists a polynomial time algorithm that provides a $(u/8, p/2)$-maxmin strategy.*

The more technically involved result of Section 4.3.2 concerns the case where the weights are not necessarily uniform. We show a reduction from the case of non-uniform weights to the case of uniform weights that loses an $O(\log K)$ on the payoff of the algorithm. The high-level idea is as follows: In order to approximate a (u, p)-maxmin strategy, we separate the battlefields into two categories *high-value* and *low-value*. High-value battlefields have a weight of at least $u/O(\log K)$ and the payoff of the low-value battlefields is below this threshold. The idea is that in order for player A to obtain a payoff of at least $u/O(\log K)$ it only suffices to win a high-value battlefield. Moreover, if the number of low-value battlefields is considerable, player A may distribute her troops over those battlefields and obtain

a payoff of at least $u/O(\log K)$. Since any high-value battlefield provides a payoff of at least $u/O(\log K)$, we can ignore the weights and play on these battlefields as if all their weights were equal. For the low-value battlefields, on the other hand, we take advantage of the fact that any battlefield of this type contributes a small payoff to the optimal solution and thus via a combinatorial argument we reduce the problem to the case of uniform weight. We then state that if player A flips a coin and plays on each set of battlefield with probability $1/2$ she can obtain a payoff of at least $u/O(\log K)$ with probability at least $p/O(1)$ given that there exists a (u, p)-maxmin strategy for player A. This method is similar to the *core-tail decomposition* technique used in [73–76] to design approximately optimal mechanisms in the worst case scenarios.

Lemma 59 [restated]. *Given that a (u, p)-maxmin strategy exists for player A in an instance of discrete Colonel Blotto, there exists a polynomial time algorithm that provides a $(u/(16(\lceil \log K \rceil + 1)), p/4)$-maxmin strategy.*

We also consider the continuous Colonel Blotto problem in Section 4.4. In the continuous variant of the problem, the players may allocate a real number of troops to a battlefield. We show that this enables us to exactly compute the optimal guaranteed payoff of player A with an LP. Recall that the best response of player B in the guaranteed payoff case can be modeled via a knapsack problem. Let a_i denote the number of troops that player A allocates to a battlefield i. Based on the knapsack model, player B's best response is to select a subset S of battlefields with the

maximum possible payoff subject to $\sum_{i \in S} a_i \leq M$. Indeed this is a linear constraint and thus the problem of computing a $(u, 1)$-maxmin strategy can be formulated as a linear program as follows: define K variables $a_1, a_2, \ldots, a_K$ to denote the number of troops of player A in each of the K battlefields. Every strategy of player B cannot get a payoff more than $\sum w_i - u$ and thus any subset of battlefields with a total weight of at least $\sum w_i - u$ should have a total number of troops more than M. Of course, this adds an exponential number of linear constraints to the LP, nonetheless, we show that ellipsoid method can solve this program in polynomial time.

Lemma 60 [restated]. For any given instance of continuous Colonel Blotto and any given u, there exists a polynomial time algorithm to either find a $(u, 1)$-maxmin strategy or report that no $(u, 1)$-maxmin strategy exists.

Furthermore, similar to the high-value, low-value decomposition of the battlefields we described above, we show that the problem for the case of (u, p)-maxmin reduces to the case of uniform battlefield weights and as a result, one can design a polynomial time algorithm to provide a constant approximation of a (u, p)-maxmin strategy.

Theorem 62 [restated]. Given that a (u, p)-maxmin strategy exists for player A in an instance of continuous Colonel Blotto, Algorithm 8 provides a $(u/8, p/8)$-maxmin strategy.

Finally, in Section 4.5 we study the notion of (u, p)-maxmin strategies for the

auditing game. In the auditing game, an instance of the Colonel Blotto game (such as the US presidential election) is given and a hacker is trying to meddle in the game in favor of one of the players, say player A. Therefore, each strategy of the hacker is to choose a subset of the battlefields in which player A loses and flip the results of those battlefields by hacking the system. The auditor, on the other hand, wants to secure the game by establishing extra security for up to m battlefields. If the auditor protects a battlefield that the hacker attacks, she'll catch the attacker and thus the attacker receives a payoff of 0. Otherwise, the payoff of the hacker is the total sum of the weights of the states that she hacks. The game is constant-sum and the summation of the payoffs of the players is always the total number of electoral votes. Note that both the auditor and the hacker are aware of the strategies in the Colonel Blotto instance.

In Section 4.5, we seek to approximate a (u, p)-maxmin strategy for the auditor in this game. We show that given that there exists a (u, p)-maxmin strategy for the auditor, one can find in polynomial time a strategy for the auditor which is at least $(u, p/(1 - 1/e))$-maxmin. To this end, we define a benchmark LP and make a connection between the optimal solution of this LP and the highest probability for which the auditor can obtain a payoff of at least u. Next, we take the dual of the program and based on a primal-dual argument, provide a strategy for the auditor that guarantees a payoff of at least u with at least a probability of p'. Finally, we make a connection between p' and the solution of the benchmark LP and argue that p' is at least a $1 - 1/e$ fraction of p. This yields the following theorem.

Theorem 64 [restated]. For any given u and p, where a (u, p)-maxmin strategy is guaranteed to exist for the auditor, a polynomial time algorithm exists that finds a $(u, (1 - 1/e)p)$-maxmin approximation strategy.

Last but not least, we show a reduction from the auditing game to an instance of the Colonel Blotto game when the winner of each battlefield is specified by a given function.

4.1 Preliminaries

Colonel Blotto Game In the Colonel Blotto game, two players A and B are competing over a number of battlefields. We denote the number of battlefields by K and denote by N and M the total troops of players A and B, respectively. Associated to each battlefield i is a weight w_i which shows the amount of profit a player wins if she wins that battlefield. This way, every strategy of a player is a partitioning of her troops over the battlefields. In the *discrete* version of Colonel Blotto, the number of troops that the players put in the battlefields must be an integer. In contrast, in the *continuous* version, a battlefield may contain any fraction of the troops. Player A wins a battlefield i if she puts more troops in that battlefield than her opponent. For simplicity, we break the ties in favor of player B, that is, if player B puts as many troops as player A's troops in a battlefield i, then she wins that battlefield and receives a payoff of w_i. The final payoff of the players in this game is the total payoff she receives over all battlefields. For a pair of pure strategies x and

y, we denote by $\mathsf{u}^{\mathsf{A}}(x,y)$ and $\mathsf{u}^{\mathsf{B}}(x,y)$ the payoff of the players if they play x and y respectively. Similarly, for a pair of mixed strategies X and Y we have

$$\mathsf{u}^{\mathsf{A}}(X,Y) = \mathop{\mathbb{E}}_{x\sim X, y\sim Y}[\mathsf{u}^{\mathsf{A}}(x,y)] \qquad\qquad \mathsf{u}^{\mathsf{B}}(X,Y) = \mathop{\mathbb{E}}_{x\sim X, y\sim Y}[\mathsf{u}^{\mathsf{B}}(x,y)].$$

Auditing game. Suppose there are K states in a presidential election race, each corresponding to a number of electoral votes. An outsider wants to hack into the system and change the outcome of the election in favor of the losing candidate. Moreover, an auditor wants to make recounts to avoid possible frauds. Refering to the players by the *hacker* and the *auditor*, we assume the simplest and full information case, that is both the hacker and the auditor have access to the exact results.[1] If the auditor conducts an inspection in a state whose winner is manipulated by the hacker, she catches the hacker and thus she wins the game (i.e., receives the maximum possible utility). On the other hand, if the hacker survives the inspection, her utility would be the number of electoral votes that she hacks in favor of her candidate.

We formally define the game as follows. Given in the input, is a set $\{s_1, s_2, \ldots, s_K\}$ of K states. For each state s_i, a value v_i is specified in the input, which is the number of its electoral votes if it is won by the losing candidate (i.e., the hacker's candidate) and zero otherwise. A limit m on the number of states that can be inspected by the auditor is also given in the input. A strategy of the hacker is a subset H of the states to hack, and a strategy of the auditor is a set A of size at most m of the states

[1] In practice, this could be obtained by polls.

to audit. The game is constant sum and the summation of utilities is always $\sum v_i$. If the attacker is caught (i.e., if $H \cap A \neq \emptyset$), the auditor receives utility $\sum v_i$ and the attacker receives utility 0. However, if the attacker is not caught (i.e., if $H \cap A = \emptyset$), she receives utility $\sum_{s_i \in H} v_i$ and the auditor receives utility $\sum v_i - \sum_{s_i \in H} v_i$.

Similar to the notation that we use for the Colonel Blotto problem, for (possibly mixed) strategies x and y, we denote by $\mathsf{u}^{\mathsf{A}}(x, y)$ and $\mathsf{u}^{\mathsf{B}}(x, y)$ the payoff of the auditor and the hacker if they play x and y respectively.

(u, p)-maxmin strategies. We call a strategy of a player, a (u, p)-maxmin, if it guarantees a utility of at least u for her with probability at least p, regardless of her opponent's strategy. In other words, a strategy x is (u, p)-maxmin if for every (possibly mixed) strategy y of the opponent we have

$$\Pr_{x \sim X, y \sim Y}[\mathsf{u}^{\mathsf{A}}(x, y) \geq u] \geq p.$$

4.2 Maximizing Expectation Versus (u, p)-maxmin Strategies

In this section, we compare algorithms that are designed to maximize the expected payoff to the algorithms that are specifically designed to approximate a (u, p)-maxmin strategy from two perspectives: (i) the approximation factor that they guarantee; (ii) their computational complexity.

4.2.1 Comparison of the Approximation Factors

As it was already mentioned, our main results are algorithms that approximate the problem of finding a (u, p)-maxmin strategy. Given that at least for the Colonel Blotto problem, exact algorithms that maximize the expected payoff exist [1, 2], one might be interested in the possible approximation factor that an expectation maximizer algorithm guarantees, and whether designing new algorithms is really needed or not.

Conceptually, the main difference between an expectation maximizer algorithm and a (u, p)-maxmin optimizer, is in that the (u, p)-maxmin optimizer, in addition to the instance of the game, takes u and p as extra parameters in the input and designs a strategy accordingly, whereas the expectation maximizer returns a single strategy for the game instance. Therefore an expectation maximizer would achieve a relatively good approximation of our problem, only if it does so for all possible values of u and p.

Unsurprisingly, this is not the case. The following theorem implies that the expectation maximizers do not guarantee any constant approximation for (u, p)-maxmin problem.

Theorem 46. *For any given u, p and arbitrarily small constants $0 < \alpha < 1$ and $0 < \beta < 1$, there exists an instance of Colonel Blotto (both discrete and continuous), where for any approximate $(\alpha' u, \beta' p)$-maxmin strategy that an expectation maximizer algorithm returns, either $\alpha' < \alpha$ or $\beta' < \beta$.*

Proof. We construct a Colonel Blotto instance in such a way that guarantees ex-

istence of a (u, p)-maxmin strategy for player A. Then show that an expectation maximizer algorithm does not achieve anything strickly better than an $(\alpha u, \beta p)$-maxmin solution.

Construct the following Colonel Blotto instance: as usual, denote the troops of player A by N and assume that player B has $M = N/(\beta p) - 1$ troops.[2] The game has $K = 1/(\beta p) + M/(1-p)$ battlefields, where $1/(\beta p)$ of which are called *high-value* battlefields and the rest are called *low-value* battlefields. The weight of a high-value battlefield is a sufficiently large number which we denote by ∞ (suffices if $\infty > Nu$) and the weight of a low-value battlefield is u.

We first show that in the mentioned instance, player A has a (u, p)-maxmin strategy. The strategy is as follows: choose one low-value battlefield uniformly at random and put N troops in it. Since there are $M/(1-p)$ low-value battlefields, in any pure strategy, player B can put a non-zero number of troops in at most a $(1-p)$ fraction of the low-value battlefields. Hence player A wins a low-value battlefield with probability at least $1-(1-p) = p$. Since the low-value battlefields have weight u, this is a (u, p)-maxmin strategy.

However, if the goal of player A is to achieve the maximum expected payoff she will end up playing a totally different strategy. Since $M < N/(\beta p)$, there exists at least one high-value battlefield in which player B puts less than N troops, so the strategy that maximizes the expected payoff of player A is to choose a high-value battlefield uniformly at random and put N troops in it. This guarantees that with

[2]Assume for ease of exposition that all the fractions that are used in constructing the strategy are integers, otherwise consider a smaller value for β for which that holds.

probability at least βp, player A wins a high-value battlefield, which achieves an expected payoff of at least $\beta p \cdot \infty$. It is easy to see that no other strategy of player A achieves this expected payoff. Note that this strategy gets a non-zero payoff with probability at most βp. Hence for any $\alpha u > 0$, it does not achieve any strategy that is strictly better than $(\alpha u, \beta p)$-maxmin. □

However, we show that for the special case where a (u, p)-maxmin strategy, for sufficiently large values of u and p is guaranteed to exist, the solution of an expectation maximizer is a good approximation of it.

Lemma 47. *Let $W := \Sigma_{i=1}^{k} w_i$ denote the total weight of the battlefields. Given that there exists a (u, p)-maxmin strategy of player A where $u \geq \frac{W}{\alpha}$ and $p \geq \frac{1}{\beta}$ for $\alpha, \beta \geq 1$ the expected maximizer returns a $(\frac{u}{2\beta}, \frac{p}{2\alpha})$-maxmin strategy.*

Proof. Let U denote the maximum expected utility of player A. Since there exist a (u, p)-maxmin strategy of player A, $U \geq u \cdot p$. Note that $u \cdot p \geq \frac{W}{\alpha\beta}$, and the maximum payoff that a player achieves from playing a strategy is W. Let q denote the probability with which player A achieves more than $\frac{u}{2\beta}$ utility in the strategy with expected utility of U. Therefore, $U < (1 - q) \cdot \frac{u}{2\beta} + qW$. Assume that $q < \frac{p}{2\alpha}$ then we obtain a contradiction. In this case,

$$U < (1 - \frac{p}{2\alpha}) \cdot \frac{u}{2\beta} + \frac{p}{2\alpha} W \leq (1 - \frac{p}{2\alpha}) \frac{W}{2\alpha\beta} + \frac{W}{2\alpha\beta} < \frac{W}{\alpha\beta}$$

holds which contradicts $U \geq \frac{W}{\alpha\beta}$, so the expected maximizer strategy is a $(\frac{u}{2\beta}, \frac{p}{2\alpha})$-maxmin strategy of player A. □

4.2.2 Comparison of their Computational Complexity

Let us again focus on the Colonel Blotto problem. It has been shown in the literature that the problem of finding a maxmin strategy that maximizes the expected payoff could be efficiently solved in polynomial time [1,2]. The goal of this section is to illustrate why the problem of finding a (u,p)-maxmin strategy seems to be computationally harder.

In particular, we show that while the "best response" could easily be computed in polynomial time when the goal is to maximize the expected payoff [2], it is NP-hard to find the best-response for the case where the goal is to give a (u,p)-maxmin strategy.

Although the hardness of finding the best response does not necessarily imply any hardness result for the actual game; it is often a good indicator of how hard the game is to solve. We refer interested readers to the paper of Xu [78] which, for a large family of games, proves solving the actual game is as hard as finding the best response.

Assume a (mixed) strategy s_B of player B, and a minimum utility u are given; the best response problem for the Colonel Blotto game, denoted by $\mathcal{BR}$, is to find a pure strategy s_A of player A which maximizes $\Pr[\mathsf{u}^{\mathsf{A}}(s_A, s_B) \geq u]$.

Theorem 48. *There is no polynomial time algorithm to solve $\mathcal{BR}$ unless P=NP.*

Proof. To prove this hardness, we reduce max-coverage problem to $\mathcal{BR}$. A number k and a collection of sets $S = \{S_1, S_2, \ldots, S_n\}$ are given. The maximum coverage problem is to find a subset $S' \subseteq S$ of the collection, such that $|S'| \leq k$ and the

number of covered elements (i.e., $|\cup_{S_i \in S'} S_i|$) is maximized.

Consider an instance of Colonel Blotto game with $|S|$ battlefields of the same weight where the first player has k troops and the second player has a sufficiently large number of troops.

Let $E = \cup_{S_i \in S} S_i$ denote the set of all items in the given max coverage instance. The support of the mixed strategy s_B of player B contains $|E|$ pure strategies, each corresponding to an item and player B plays one of these pure strategies uniformly at random (i.e., all of the pure strategies are played with the same probability). For the corresponding pure strategy to an item $e \in E$, we put $k+1$ troops in each battlefield that its corresponding set does not contain e and put zero troops in all other battlefields.

Assume that our goal is to find the best response of player A that maximizes the probability of winning at least one battlefield. Let p denote this probability. We claim $p|E|$ is indeed the solution of the max-coverage instance. Since player B puts either 0 troops or $k+1$ troops in each battlefield, it suffices for player A to put either 0 or 1 troops in each battlefield. To see this recall that player A has only k troops and clearly cannot win the battlefields in which player B puts $k+1$ troops. If player A puts more than zero troops in a battlefield, we choose its corresponding set in the max-coverage problem. Since there are at most k such battlefields, it is indeed a valid solution and it is easy to see that it maximizes the number of covered elements. □

4.3 Discrete Colonel Blotto

4.3.1 Approximating $(u, 1)$-maxmin

In this section we study the problem of finding a $(u, 1)$-maxmin strategy for player A with the maximum possible u. In this case, u is also called the guaranteed payoff that player A achieves in an instance of Colonel Blotto game. It is easy to see that the maximum guaranteed payoff of player A in a Colonel Blotto game, denoted by OPT is equal to the minimax strategy in a slightly modified version of this game which is as follows: player A first chooses a pure strategy and reveals it to her opponent, then player B, based on this observation, plays a pure strategy that maximizes her payoff. In this game if $N \leq M$, there exists no strategy of player A that guarantees a payoff more than zero for this player, therefore in this section we assume that $N > M$.

Let S_A be an arbitrary pure strategy of player A, and let strategy R_B be a best response of player B to S_A. We first give an algorithm that finds a response R'_B of player B such that $\mathsf{u}^{\mathsf{B}}(S_A, R'_B) \geq \mathsf{u}^{\mathsf{B}}(S_A, R_B) - \max_{i=1}^{K} w_i$. Then, by fixing this algorithm for player B, we are able to find a strategy of player A that guarantees at least half of the maximum guaranteed payoff of player A.

Lemma 49. *Let R'_B denote the strategy that Algorithm 4 provides for player B, and let R_B denote her best response against the strategy of player A, which is denoted by S. Thus, we have $\mathsf{u}^{\mathsf{B}}(S, R'_B) \geq \mathsf{u}^{\mathsf{B}}(S, R_B) - \max_{i=1}^{K} w_i$.*

Proof. Let s_i denote the number of troops that player A puts in the i-th battlefield

Algorithm 4: An approximation algorithm for the best response of player B

1: Let w_i denote the weight of the i-th battlefield and let s_i denote the number of troops that player A has in this battlefield.
2: Sort the battlefields such that for any $i \in [K-1]$, $\frac{w_i}{s_i} \geq \frac{w_{i+1}}{s_{i+1}}$
3: $i \leftarrow 1$
4: **while** $M \geq s_i$ **do**
5: Player B puts s_i troops in the i-th battlefield
6: $M \leftarrow M - s_i$
7: $i \leftarrow i + 1$
8: **end while**

while playing strategy S, and let $r_i := w_i/s_i$ denote the *value* of a battlefield.

The algorithm first sorts the battlefields in the decreasing order of their values. Assume w.l.g. that the initial order is the desired one, i.e., $r_{i-1} \geq r_i$. Starting from the first battlefield in the sorted order, player B puts as many troops as player A has until there are no more troops left for player B. Let the k-th battlefield be the stopping point of the algorithm. Clearly $\mathsf{u}^{\mathsf{B}}(S, R'_B) = \Sigma_{i=1}^{k-1} w_i$. Moreover, one can easily see that $\mathsf{u}^{\mathsf{B}}(S, R_B) \leq \Sigma_{i=1}^{k} w_i$. Therefore $\mathsf{u}^{\mathsf{B}}(S, R'_B) \geq \mathsf{u}^{\mathsf{B}}(S, R_B) - \max_{i=1}^{K} w_i$ as desired. □

Theorem 50. *There exists a polynomial time algorithm that gives a* $(u/2, 1)$*-*maxmin *strategy of player A, assuming that* u *is the maximum guaranteed payoff of player A.*

Proof. We first define a new optimization problem, then we prove that the solution to that problem is also a 2-approximation solution for the maximum guaranteed payoff of player A (i.e., OPT). For any strategy s of player A, let $U(s)$ denote the payoff that player A achieves if the response of player B to strategy s is determined by Algorithm 4. The optimization problem is to find a strategy s of player A

that maximizes $U(s)$. Let S denote the set of all possible strategies of player A, and let $\textsc{opt}' = \max_{s \in S} U(s)$. Since this is a constant-sum game, by Lemma 49, $\textsc{opt}' \leq \textsc{opt} + \max_{i=1}^{K} w_i$. Moreover, $\textsc{opt} \geq \max_{i=1}^{K} w_i$ since $N > M$, and player A can win the battlefield with maximum weight by putting all her troops in that battlefield. Therefore, $\textsc{opt}' \leq 2\textsc{opt}$, and to prove this lemma it suffices to give an algorithm that finds $\textsc{opt}'$. Algorithm 5 finds $\textsc{opt}'$ via dynamic programming.

Algorithm 5: A 2-approximation algorithm for the guaranteed payoff of player A

```
1: function ApproximateGuaranteedPayoff
2:     Let w_k denote the weight of k-th battlefild.
3:     s ← −∞
4:     for k in {1, ..., K} do
5:         for m in {1, ..., N} do
6:             s ← max(s, FindBestPayoff(m, k))
7:         end for
8:         return s
9:     end for
10: end function
11: function FindBestPayoff(m, k)
12:     r ← w_k/m
13:     U[0][0][0] ← 0
14:     for any i in {1, ..., K}, a in {0, ..., N − m} and b in {0, ..., M} do
15:         U[i][a][b] ← −∞
16:         if i = k then
17:             U[i][a][b] ← U[i − 1][a − m][b] + w_i
18:         else
19:             for t in {0, ..., min(a, b, ⌈w_i/r⌉ − 1)} do
20:                 U[i][a][b] ← max(U[i][a][b], U[i − 1][a − t][b − t])
21:             end for
22:             if a ≥ ⌈w_i/r⌉ then
23:                 U[i][a][b] ← max(U[i][a][b], U[i − 1][a − ⌈w_i/r⌉][b] + w_i)
24:             end if
25:         end if
26:     end for
27:     return max_{i=0}^{m−1} U[K][N][M − i]
28: end function
```

For any strategy s of player A, let s_i denote the number of troops that player

A puts in the i-th battlefield and let $B(s)$ denote the set of battlefields that player A wins given that the response of player B is determined by Algorithm 4. In addition let $b(s) := \arg\max_{i \in B(s)} \frac{w_i}{s_i}$ be the first battlefield in which player B loses (in the sorted list of battlefields in Algorithm 4) and let $t(s) := s_{b(s)}$. Furthermore, let $S(k, m)$ be a subset of strategies of player A where $b(s) = k$ and $t(s) = m$ for any $s \in S(k, m)$. Function FindBestPayoff, for given inputs $1 \leq k \leq K$ and $0 \leq m \leq N$ finds $\max_{s \in S(k,m)} U(s)$. Finally, in function ApproximateGuaranteedPayoff, we find OPT$'$ by calling function FindBestPayoff for all sets $S(k, m)$ such that $1 \leq k \leq$ and $0 \leq m \leq N$, and returning the maximum answer.

Let $U^A(j, s)$ denote the payoff that player A achieves in the j-th battlefield if the response of player B to strategy s is determined by Algorithm 4, and let $P(i, a, b)$ denote the set of strategies of player A such that for any $s \in P(i, a, b)$, $\Sigma_{j=1}^{i} s_j = a$, and in the response to s that is determined by Algorithm 4, player B puts exactly b troops in the first i battlefields.

Claim 51. *In Algorithm 5,*

$$U[i][a][b] = \max_{s \in S(k,m) \cap P(i,a,b)} \Sigma_{j=1}^{i} U^A(j, s).$$

Proof. We use induction on i.

1. Induction hypothesis: for any $1 \leq i \leq K$, and any arbitrary a' and b' such that $0 \leq a' \leq N$ and $0 \leq b' \leq M$, $U[i][a'][b'] = \max_{s \in S(k,m) \cap P(i,a',b')} \Sigma_{j=1}^{i} U^A(j, s)$.

2. Base case: $U[0][0][0] = 0$ and all the other cells for $i = 0$ are undefined (we

assume that the value of any undefined cell is equal to $-\infty$).

3. For the induction step we prove the correctness of hypothesis for $i+1$: It is easy to verify if $i+1=k$ since by the constraints, player A puts exactly m troops and player B puts no troops in the k-th battlefield, therefore $U[i][a][b] = U[i-1][a-m][b]+w_i$. However, if $i+1 \neq k$, there are different possible cases for the number of troops that player A puts in this battlefield, denoted by s_i. As a result she either gains w_{i+1} or 0 utility in this battlefield. By Algorithm 4, if $\frac{w_{i+1}}{s_{i+1}} \geq \frac{w_k}{m}$ player B wins this battlefield, otherwise she loses it. In other words, if $s_{i+1} < \frac{m \times w_{i+1}}{w_k}$, by Algorithm 4, player B, puts s_{i+1} troops in this battlefield and wins it. These cases are handled in line 20 of the function FindBestPayoff. In addition, if $s_{i+1} \geq \frac{m \times w_{i+1}}{w_k}$, by Algorithm 4, player B puts no troop in it so player A wins it. Also, player A never puts more than $\lceil \frac{m \times w_{i+1}}{w_k} \rceil$ in this battlefield since any number of troops more than this results in the same payoff (winning w_{i+1}). This case is handled in line 23 of the algorithm. To sum up, the induction step is proved.

□

To complete the proof, it suffices to prove that function FindBestPayoff correctly finds $\max_{s \in S(k,m)} U(s)$. Note that the output of function FindBestPayoff is $\max_{i=0}^{m-1} U[K][N][M-i]$, hence we need to prove

$$\max_{s \in S(k,m)} U(s) = \max_{i=0}^{m-1} U[K][N][M-i]. \tag{4.1}$$

Claim 51 is indeed the main technical ingredient that we use to prove (4.1).

$$\max_{i=0}^{m-1} U[K][N][M-i] = \max_{i=0}^{m-1} \Big(\max_{s \in S(k,m) \cap P(K,A,B-i)} \Sigma_{j=1}^{K} U^A(j,s) \Big) \qquad \text{By Claim 51}$$

(4.2)

$$= \max_{i=0}^{m-1} \Big(\max_{s \in S(k,m) \cap P(K,A,B-i)} U(s) \Big) \tag{4.3}$$

$$= \max_{s \in S(k,m) \cap (\cup_{i=0}^{m-1} P(K,A,B-i))} U(s) \tag{4.4}$$

Note that player B may have at most $m-1$ unused troops since otherwise she could use them to win battlefield k which contradicts the assumption of this function. This implies $S(k,m) \subseteq \cup_{i=0}^{m-1} P(K,A,B-i)$ and therefore by (4.4) we obtain (4.1) as desired. □

4.3.2 Approximating (u,p)-maxmin

In this section, we present a polynomial time algorithm for approximating a (u,p)-maxmin strategy in the Colonel Blotto game. More precisely, given that there exists a (u,p)-maxmin strategy for player A, we present a polynomial time algorithm to find a $(O(u/(\log K)), O(p))$-maxmin strategy. In Section 4.3.2.1 we study the problem for a special case where $w_i = 1$ for all $i \in [K]$. We show that in this case, if K and N are large enough then player A can win a fraction of the battlefields proportional to the ratio of N over M. We also argue that in some cases, no strategy can be (u,p)-maxmin for player A with $u,p > 0$. We then use these observations to obtain a $(u/8, p/4)$-maxmin strategy for player A in the uniform setting and a

$(u/(16(\lceil \log K \rceil + 1)), p/8)$-maxmin strategy for the general setting.

4.3.2.1 The Case of Uniform Weights

A special case of the problem is when all weights are uniform. We study this case in this section. We assume w.l.g. that all weights are equal to 1 since one can always satisfy this condition by scaling the weights. Given that there exists a (u, p)-maxmin strategy for player A, we present a strategy for player A that is at least $(u/8, p/4)$-maxmin. Recall that we denote the number of troops of players A and B by N and M, respectively. Our first observation is that if both u and p are non-zero then $p \leq 4(K - \lfloor M/N \rfloor)/K$ holds.

Lemma 52. *Given that there exists a (u, p)-maxmin strategy for player A with $u, p > 0$, then $p \leq 2(K - \lfloor M/N \rfloor)/K$ holds.*

Proof. We assume w.l.g. that $\lfloor M/N \rfloor \geq 1$ (otherwise $p \leq 2(K - \lfloor M/N \rfloor)/K = 2$ trivially holds). Also $K \geq 2\lfloor M/N \rfloor$ implies $2(K - \lfloor M/N \rfloor)/K \geq 1$ which yields $p \leq 2(K - \lfloor M/N \rfloor)/K$. Suppose for the sake of contradiction that the conditions doesn't hold. Therefore we have $K < 2\lfloor M/N \rfloor$ and also $p > 2(K - \lfloor M/N \rfloor)/K$. We show that in this case, no strategy of player A can be (u, p)-maxmin for $u > 0$.

If $K \leq \lfloor M/N \rfloor$ then player B can put N troops in all battlefields and always prevent player A from winning any battlefield. Thus, $K > \lfloor M/N \rfloor$. Now if player B plays the following strategy, the probability that player A wins a single battlefield is smaller than p: randomly choose $2(K - \lfloor M/N \rfloor)$ battlefields and put $N/2$ ($= \lfloor N/2 \rfloor$ in the discrete version of the game) troops in them and put N troops in the

rest of the battlefields. Recall that $K < 2\lfloor M/N \rfloor$ and thus $2(K - \lfloor M/N \rfloor)$ does not exceed the number of battlefields. This requires at most the total number of

$$\begin{aligned} 2(K - \lfloor M/N \rfloor)N/2 + (2\lfloor M/N \rfloor - K)N &\leq \lfloor M/N \rfloor(2N - 2N/2) + K(2N/2 - N) \\ &= \lfloor M/N \rfloor(2N - N) + K(N - N) \\ &= \lfloor M/N \rfloor N \\ &\leq M \end{aligned}$$

troops. Notice that in order for a strategy of player A to win a battlefield, it needs to put more than $N/2$ troops in that battlefield. Moreover, each pure strategy of player A can put more than $N/2$ troops in at most one battlefield. Thus, a pure strategy of player A gains a non-zero payoff only if player B puts at most $N/2$ troops in that chosen battlefild. This probability is bounded by $2(K - \lfloor M/N \rfloor)/K$ for each battlefield due to the strategy of player B. Therefore, player A can get a non-zero payoff with probability no more than $2(K - \lfloor M/N \rfloor)/K$. This contradicts the existence of a (u, p)-maxmin strategy for player A with $u > 0$ and $p > 2(K - \lfloor M/N \rfloor)/K$. □

Although we consider Lemma 52 in the uniform and discrete setting, the proof doesn't rely on any of these conditions. Thus, Lemma 52 holds for the general setting (both continuous and discrete). Based on Lemma 52, we present a simple algorithm and show that the strategy obtained from this algorithm is at least $(u/8, p/4)$-maxmin. In our algorithm, if $K \leq 2\lfloor M/N \rfloor$ then we randomly select a battlefield and put N troops in it. Otherwise, we find the smallest t such that $t(\lfloor K/2 \rfloor + 1) > M$

and put t troops in $\lfloor N/t \rfloor$ battlefields uniformly at random. The logic behind this is that we choose a large enough t to make sure player B can put t troops in no more than $\lfloor K/2 \rfloor$ battlefields. Therefore, when player A puts t troops in a random battlefield, we can argue that she wins that battlefield with probability at least $1/2$ regardless of player B's strategy. We use this fact to show that player A wins at least $\lceil 1/8 \min\{N, K, K(N/M)\} \rceil$ battlefields with probability at least $1/2$. Finally we provide almost matching upper bounds to show the tightness of our solution. We first provide a lower bound in Lemma 53 on the payoff of this strategy against any response of player B.

Algorithm 6: An algorithm to find a $(u/8, p/4)$-maxmin strategy for player A

---

if $K < 2\lfloor M/N \rfloor$ **then**
    Choose a battlefield i uniformly at random.
    Put N troops in battlefield i.
else
    $t \leftarrow 0$.
    while $t(\lfloor K/2 \rfloor + 1) \leq M$ **do**
        $t \leftarrow t + 1$
    end while
    if $N \geq Kt$ **then**
        put t troops in all battlefields
    else
        Choose $\lfloor N/t \rfloor$ battlefields $a_1, a_2, \ldots, a_{\lfloor N/t \rfloor}$ uniformly at random.
        Put t troops in every battlefield a_i.
    end if
end if

Lemma 53. *The strategy of Algorithm 6 provides the following guarantees for player A in any instance of discrete Colonel Blotto game:*

- *If $K < 2\lfloor M/N \rfloor$, player A wins a battlefield with probability at least $(K - \lfloor M/N \rfloor)/K$.*

- *If* $K \geq 2\lfloor M/N \rfloor$*, player A wins* $\lceil 1/8 \min\{N, K, K(N/M)\} \rceil$ *battlefields with probability at least* $1/2$*.*

Proof. We prove each of the cases separately. If $K < 2\lfloor M/N \rfloor$, player A's strategy is to randomly choose a battlefield and put all her troops in it. Notice that any strategy of player B can put N troops in at most $\lfloor M/N \rfloor$ battlefields. Thus, with probability at least $(K - \lfloor M/N \rfloor)/K$, player B puts fewer than N troops in the selected battlefield of player A and thus player A wins that battlefield.

If $K \geq 2\lfloor M/N \rfloor$, Algorithm 6 finds the smallest t such that $t(\lfloor K/2 \rfloor + 1) > M$ and puts t troops in $\min\{K, \lfloor N/t \rfloor\}$ randomly selected battlefields. As we mentioned earlier, since $t(\lfloor K/2 \rfloor + 1) > M$, player B can put t troops in no more than $\lfloor K/2 \rfloor$ battlefields and thus she puts fewer than t troops in at least half of the battlefields. If $K \leq \lfloor N/t \rfloor$, player A puts t troops in all battlefields and since player B can protect at most $\lfloor K/2 \rfloor$ of the battlefields, player A wins at least $\lceil K/2 \rceil$ battlefields with probability 1. If $K > \lfloor N/t \rfloor$, player A wins any of the selected battlefields with pobability at least $1/2$ and therefore, with probability at least $1/2$, player A wins at least $\lceil \lfloor N/t \rfloor /2 \rceil$ of the $\lfloor N/t \rfloor$ battlefields wherein she puts t troops. The rest of the proof follows from a mathematical observation. In the interest of space we omit the proof of Observation 54 here.

Observation 54. *Let* N*,* M*, and* K *be three positive integer numbers such that* $K \geq 2\lfloor M/N \rfloor$ *and* t *be the smallest integer number such that* $t(\lfloor K/2 \rfloor + 1) > M$*. Then we have*

$$\lceil \lfloor N/t \rfloor /2 \rceil \geq \lceil 1/8 \min\{N, K(N/M)\} \rceil.$$

□

To show that Algorithm 6 provides a strategy competitive to that of the optimal, we present two upper bounds for each of the cases separately.

Lemma 55. *Given that there exists a (u, p)-maxmin strategy for player A with non-zero u and p in an instance of discrete Colonel Blotto with uniform wrights, for $K < 2\lfloor M/N \rfloor$ we have $u \leq 2$ and $p \leq 2(K - \lfloor M/N \rfloor)/K$.*

Proof. $p \leq 2(K - \lfloor M/N \rfloor)/K$ follows directly from Lemma 52. Next we argue that in this case u is also bounded by 2. To this end, suppose that player B puts $\lfloor M/K \rfloor$ troops in every battlefield. This way, in order for player A to win a battlefield, she should put at least $\lfloor M/K \rfloor + 1 \geq \lceil M/K \rceil$ troops in that battlefield. Since $K < 2\lfloor M/N \rfloor$, then $\lceil M/K \rceil \geq N/2$ and thus player A can never achieve a payoff more than 2. □

Lemma 56. *Given that there exists a (u, p)-maxmin strategy for player A with non-zero u and p in an instance of discrete Colonel Blotto with uniform wrights, for $K \geq 2\lfloor M/N \rfloor$ we have $u \leq \min\{K, N, K(N/M)\}$.*

Proof. $u \leq K$ and $u \leq N$ hold since there are at most K battlefields to win and player A can put non-zero troops in at most N of them. Therefore, the only non-trivial part is to show $u \leq K(N/M)$. We show that no strategy of player A can achieve a payoff more than $K(N/M)$ with non-zero probability. To this end, suppose that player B puts $\lfloor M/K \rfloor$ troops in every battlefield. This way, in order for player A to win a battlefield, she has to put at least $\lfloor M/K \rfloor + 1 \geq \lceil M/K \rceil$ troops in

that battlefield. Thus, any strategy of player A wins no more than $N/\lceil M/K \rceil \leq K/(M/N)$ battlefields. Therefore, u is bounded by $K/(M/N)$. □

Lemma 53 along with the upper bounds provided in Lemmas 55 and 56 prove that the strategy of Algorithm 6 is competitive with the optimal strategy of player A.

Theorem 57 (as a corollary of Lemmas 53, 55, and 56)**.** *Given that there exists a* (u, p)-*maxmin strategy for player A in an instance of discrete Colonel Blotto with uniform weights, Algorithm 6 provides a* $(u/8, p/2)$-*maxmin strategy.*

4.3.2.2 The General Setting

We showed in Section 4.3.2.1 that when all the weights are equal to 1, there is a polynomial time solution for finding an approximate (u, p)-maxmin solution for a given u and p. In this section, we extend this result to the case of non-uniform weights. The main ingredient of our proposal is a mathematical argument which we state in Lemma 58.

Lemma 58. *Given* n *non-negative values* $a_1 \geq a_2 \geq a_3 \geq \ldots \geq a_n$*, there exists a* k *with* $1 \leq k \leq n$ *such that*

$$ka_k \geq 1/(\lceil \log n \rceil + 1) \sum_{i=1}^{n} a_i.$$

Proof. We assume w.l.g. that $n = 2^k - 1$ for some k (otherwise we add enough 0's to the end of the sequence to satisfy this condition). We divide the sequence into

$\lceil \log n \rceil$ buckets as follows: The first bucket contains only a_1. The second bucket contains a_2 and a_3. More precisely, the i'th bucket contains all variables from $a_{2^{i-1}}$ to a_{2^i-1}. Since the variables are non-decreasing, for each bucket i the summation of the variables inside it is upper bounded by $2^{i-1}a_{2^{i-1}}$. Since we have $\lceil \log n \rceil$ buckets, the total sum of the values of at least one bucket is no less than $(\sum a_i)/\lceil \log n \rceil$ and therefore at least for one i we have $2^{i-1}a_{2^{i-1}} \geq (\sum a_i)/\lceil \log n \rceil$. An extra $+1$ appears in the guarantee because of the adjustment to n that we made in the beginning of the proof. □

Given that player A has a (u, p)-maxmin strategy, we present a randomized strategy for player A and show that this strategy is at least $(u/(16(\lceil \log K \rceil + 1)), p/8)$-maxmin. In our strategy, we split the battlefields into two categories *high-value* and *low-value*. A battlefield is called high-value if the winner of that battlefield obtains a payoff of at least $u/(16(\lceil \log n \rceil + 1))$ and low-value otherwise. We denote the high-value battlefields by $A = \{a_1, a_2, \ldots, a_{|A|}\}$ and the low-value battlefields by $B = \{b_1, b_2, \ldots, b_{|B|}\}$. We assume w.l.g. that both a_i's and b_i's are sorted in non-decreasing order according to the weights of the battlefields. In other words, $w_{a_1} \geq w_{a_2} \geq \ldots \geq w_{a_{|A|}}$ and $w_{b_1} \geq w_{b_2} \geq \ldots \geq w_{b_{|B|}}$. Let M' be the smallest number of troops that player B needs to put in the high-value battlefields to make sure player A wins any of such battlefields with probability less than p due to the upperbound of Lemma 52 . In our proposal, with probability 1/2 we play Algorithm 6 on the high-value battlefields with the assumption that player B has $M' - 1$ troops. Also, with probability 1/2 we play Algorithm 6 on a prefix of battlefields

$b_1, b_2, \ldots, b_k$ as if player B had $M - M'$ troops. Note that in both cases, we assume that the weights of all battlefields are equal to 1 when using Algorithm 6. If any of A or B is empty, we only play on the non-empty set. If $M' > M$, we only play on battlefields of set A. A formal description of our proposal is given in Algorithm 7.

Algorithm 7: An algorithm to find a $(u/(16(\lceil \log K \rceil + 1)), p/4)$-maxmin strategy for player A

$A = \{a_1, a_2, \ldots, a_{|A|}\} \leftarrow$ the set of battlefield with weight at least $u/(16(\lceil \log n \rceil + 1))$
$B = \{b_1, b_2, \ldots, b_{|B|}\} \leftarrow$ the set of battlefield with weight less than $u/(16(\lceil \log n \rceil + 1))$
$M' \leftarrow 0$
while $2(|A| - \lfloor M'/N \rfloor)/|A| \geq p$ **do**
    $M' \leftarrow M' + 1$
end while
$coin \leftarrow$ either 0 or 1 with equal probability
if $coin = 0$ and $|A| = 0$ **then**
    $coin \leftarrow 1$
end if
if $coin = 1$ and ($|B| = 0$ or $M' > M$) **then**
    $coin \leftarrow 0$
end if
if $coin = 0$ **then**
    Run Algorithm 6 on the battlefields $a_1, a_2, \ldots, a_{|A|}$ with N and $M' - 1$ troops for the players.
else
    if $N \geq M - M'$ **then**
        $k \leftarrow \arg\max_{i=1}^{\min\{|B|, N\}} iw_{b_i}$
    else
        $k \leftarrow \arg\max_{i=1}^{\min\{|B|, M-M'\}} iw_{b_i}$
    end if
    Run Algorithm 6 on the battlefields $b_1, b_2, \ldots, b_k$ with N and $M - M'$ troops for the players.
end if

Our claim is that Algorithm 7 is $(u/(16(\lceil \log K \rceil + 1)), p/8)$-maxmin. Before we provide a formal proof, we mention the high level idea briefly. Notice that in Algorithm 7 we flip a coin and attack each set of the battlefields with probability 1/2.

The best response of player B is always a pure strategy. Such a pure strategy either puts fewer than M' troops in the high-value battlefields or no more than $M - M'$ troops in the low-value battlefields. In each case, we argue that the strategy of Algorithm 7 performs well with probability at least $p/4$.

Theorem 59. *Given that a (u, p)-maxmin strategy exists for player A in an instance of discrete Colonel Blotto, Algorithm 7 provides a $(u/(16(\lceil \log K \rceil + 1)), p/4)$-maxmin strategy.*

Proof. In order to show that the strategy obtained from Algorithm 7 is $(u/(16(\lceil \log K \rceil + 1)), p/4)$-maxmin, we show that it achieves a payoff at least $(16(\lceil \log K \rceil + 1))$ with probability at least $p/4$ against any pure strategy of player B. To this end, we consider two cases. Either the pure strategy of player B puts fewer than M' troops in the high-value battlefields, or puts no more than $M - M'$ troops in the low-value battlefields. We investigate each of the possibilities in the following:

Fewer than M' troops in the high-value battlefields: Line 4 of Algorithm 7 terminates right after $2(|A| - \lfloor M'/N \rfloor)/|A| < p$ happens. Therefore, for $M' - 1$ troops we have $2(|A| - \lfloor (M' - 1)/N \rfloor)/|A| \geq p$. It follows from Lemma 53 that if player B puts $M' - 1$ (or fewer) troops in these battlefields and player A plays according to Algorithm 6, she wins at least a battlefield with probability at least $\min\{1/2, (|A| - \lfloor (M' - 1)/N \rfloor)/|A|\}$ which is at least $p/2$. Moreover, the payoff she achieves from winning any of the battlefields is at least $u(16(\lceil \log K \rceil + 1))$. Since the strategy of Algorithm 7 plays on the high-value battlefields with probability (at least) $1/2$, this guarantees a payoff of $u/(16(\lceil \log K \rceil + 1))$ with probability at least

$p/4$.

No more than $M - M'$ troops in the low-value battlefields: We first provide some lower bounds on summation of the weights of the low-value battlefields. We show that unless the total weight of certain battlefields is lower bounded by fixed values, player B can play in a way to prevent player A from obtaining a payoff of at least u with probability at least p. Recall that due to Line 4 of Algorithm 7, $2(|A| - \lfloor M'/N \rfloor)/|A| < p$ holds. It follows from Lemma 52 that player B can put M' troops in the high-value battlefields to make sure player A wins no high-value battlefield with probability at least $1 - p$. Therefore, efficient alloctations of the remaining $M - M'$ troops of player B to the low-value battlefields imply:

- If $N \leq |B|$ then $w_{b_1} + w_{b_2} + \ldots + w_{b_N} \geq u$ since a (u, p)-maxmin stratey of the player A should obtain at least a payoff of u from the low-value battlefields.

- $w_{b_1} + w_{b_2} + \ldots + w_{b_{|B|}} \geq u(M - M')/N$ since otherwise the following strategy of player B can prevent player A from achiving a payoff of u with non-zero probability: Let $W = \sum w_{b_i}$ be the summation of the weights of the low-value battelfields. Put $\lfloor (M - M')w_{b_i}/W \rfloor$ troops in every battlefield b_i. Note that this requires no more than $M - M'$ troops since

$$\sum (M - M')w_{b_i}/W = (M - M')(\sum w_{b_i})/W = (M - M')W/W = M - M'.$$

  In addition to this, in order for player A to win a battlefield b_i, she has to put at least $\lfloor (M - M')w_{b_i}/W \rfloor + 1 \geq \lceil (M - M')w_{b_i}/W \rceil$ troops in that battlefield.

Therefore, the ratio of the payoff over the number of necessary troop to win for each battlefield is at least $W/(M - M')$ and thus player A can obtain no more than $WN/(M - M')$ payoff. This implies $W \geq u(M - M')/N$ provided that there exists a (u, p)-maxmin strategy for player A.

- If $N \leq M - M' \leq |B|$ then $w_{b_1} + w_{b_2} + \ldots + w_{b_{|M-M'|}} \geq u(M - M')/N$: This follows from an argument similar to the one just stated. Suppose for the sake of contradiction that $w_{b_1} + w_{b_2} + \ldots + w_{b_{|M-M'|}} < u(M - M')/N$. We define $W = \sum_{i=1}^{M-M'} w_{b_i}$ and propose the following strategy for player B. For each $1 \leq i \leq M - M'$, put $\lfloor (M - M')w_{b_i}/W \rfloor$ troops in battlefield b_i and put no troops in the rest of the low value battlefields. This way, for each such battlefield the ratio of payoff per troop necessary to win that battlefield is bounded by $W/(M - M')$ for player A. Moreover, since we assume the weights of the battlefields are non-decreasing, we have $w_{b_i} \leq W/(M - M')$ for $i > M - M'$ and thus winning each of those battlefields has a payoff of at most $W/(M - M')$ for player A. Therefore, no strategy of player A can achieve a payoff more than $NW/(M - M')$ against such a strategy of player B. This implies that $w_{b_1} + w_{b_2} + \ldots + w_{b_{|M-M'|}} \geq u(M - M')/N$ provided that there exists a (u, p)-maxmin strategy for player A in this game.

The above lower bounds along with Lemma 58 imply that

$$k w_{b_k} \geq u \max\{1, (M - M')/N\}/(\lceil \log |B| \rceil + 1).$$

Moreover, $k < 2\lfloor (M - M')/N \rfloor$ cannot hold since the weight of each low-value battlefield is bounded by $u/(16\lceil \log K \rceil + 1)$. Thus, if player A plays Algorithm 6 on battlefields $b_2, b_2, \ldots, b_k$ she wins at least $\lceil 1/8 \min\{N, k, k(N/(M - M'))\} \rceil$ of them with probability at least $1/2$ due to Lemma 53. This provides player A with a payoff of at least $u/(16(\lceil \log K \rceil + 1))$ against any pure strategy of player B that puts no more than $M - M'$ troops in the low-value battlefields. Since Algorithm 7 plays on the low-value battlefields with probability at least $1/2$, this guarantees a payoff of $u/(16(\lceil \log K \rceil + 1))$ with probability at least $1/4$. □

4.4 Continuous Colonel Blotto

In this section we study the *continuous* version of the Colonel Blotto game. In this version we relax the assumption that the number of troops that a player puts in a battlefield is an integer. In fact, for certain applications (e.g., where money is the resource that is to be distributed among battlefields) the continuous model is more realistic.[3] We first show in Section 4.4.1 that it is possible to find a $(u, 1)$-maxmin strategy for each player in the continuous Colonel Blotto in polynomial time. We also give an approximation algorithm for (u, p)-maxmin in the continuous version of the game in Section 4.4.2. Our algorithm provides a $(u/8, p/8)$-maxmin strategy for any instance of continuous Colonel Blotto, given that there exists a (u, p)-maxmin for that instance.

[3]To remain consistent to the rest of the thesis, we use "troops" to refer to the resources even for the continuous version of the game.

4.4.1 An Exact Algorithm for $(u, 1)$-maxmin

In this section we provide a polynomial time algorithm to find a $(u, 1)$-maxmin strategy for player A. The formal statement of the theorem is as follows.

Theorem 60. *For any given instance of continuous Colonel Blotto and any given u, there exists a polynomial time algorithm to either find a $(u, 1)$-maxmin strategy or report that no $(u, 1)$-maxmin strategy exists.*

The algorithm is a linear program. It is worth mentioning that using this LP, one can search over u to find the maximum payoff that can be guaranteed for player A (i.e., her pure maxmin strategy).

Let $W := \sum_{i=1}^{k} w_i$ denote the total weight of all battlefields. We define a subset $S = \{b_1, \ldots, b_k\}$ of the battlefields to be *critical* if the total weight of the battlefields in it is more than $W - u$ (i.e., $\sum_{i \in S} w_i > W - u$). The following lemma is the main observation behind the LP.

Lemma 61. *A strategy S^A is an $(u, 1)$-maxmin strategy of player A if and only if for any critical subset S of the battlefields, strategy S^A puts more than M (the total troops of player B) troops into the battlefields in S.*

Proof. Assume S^A is an $(u, 1)$-maxmin strategy of player A and assume for the sake of contradiction that there exists a critical subset S of the battlefields in which S^A does not put more than M troops. Clearly, player B is able to win any battlefield $i \in S$ by putting in it the same number of troops that S^A puts in it. This means player A is only able win the battlefields that are not in S, but since S is a critical

subset of the battlefields, the total weight of the battlefields that are not in S is less than u — which contradicts the assumption that S^A is an $(u, 1)$-maxmin strategy.

For the other direction, consider a strategy S^A that puts more than M troops in any critical subset of the battlefields, we prove S^A is indeed an $(u, 1)$-maxmin strategy. Again, for the sake of contradiction, assume this is not the case and there exists a strategy S^B of player B that gets a payoff of more than $W - u$ agains S^A. The contradiction is that the subset of battlefields that player B wins is a critical subset in which player A has put at most M troops. □

We are now ready to explain the LP. There are K variables $x_1, \ldots, x_K$ where variable x_i denotes the number of troops that we put in the i-th battlefield. Apart from the constraints that enforce these variables correspond to a valid K-partitioning of N, for each critical subset, there is a constraint that ensures the total number of troops in this subset of battlefields is more than M. Since there maybe exponentially many critical subsets, we use the ellipsoid method to solve it. The linear program is formally given as LP 1.

(LP 1)

$$
\begin{aligned}
& x_i \geq 0 \qquad \forall i : 1 \leq i \leq K \\
& x_1 + x_2 + \ldots + x_K = N \\
& \sum_{j \in S} x_j > M \qquad \forall S\text{: } S \text{ is a critical subset of battlefields}
\end{aligned}
$$

To apply the ellipsoid method, we need a separation oracle that decides whether a given assignment $\hat{x} = \langle x_1, \ldots, x_K \rangle$ is a valid solution of LP 1, and if not, finds

a violated constraint. The separation oracle first checks whether the first two constraints are violated or not, and if not runs the following instance of knapsack. The knapsack has capacity M, and for any battlefield i, there is an item with volume x_i and value w_i. Clearly, the solution of this knapsack problem is the best response of player B to the strategy of player A that corresponds to $\hat{x}$. It suffices to check whether in this best response, the battlefields that player B wins form a critical subset or not. If they do not form a critical subset, by Lemma 61, $\hat{x}$ is a valid solution of LP 1, and if they form a critical subset, the constraint that corresponds to this critical subset is violated.

4.4.2 An Approximation Algorithm for (u, p)-maxmin

In this section we present a polynomial time algorithm to find a $(u/8, p/8)$-maxmin algorithm of player A in any instance of continuous Colonel Blotto given that there exist a (u, p)-maxmin strategy of her in this instance of the game. In this algorithm we partition the battlefields to two sets of low-value and high-value battlefields. If both sets are non-empty player A either puts all her troops in high-value battlefields with probability 1/4 or she puts all the troops in low-value battlefields with probability 3/4. The strategies that she plays in any of these cases are different. Algorithm 8 contains the details of the algorithm. We also prove that this algorithm gives a $(u/8, p/8)$-maxmin strategy of player A in Theorem 62.

Theorem 62. *Given that a (u, p)-maxmin strategy exists for player A in an instance of continuous Colonel Blotto, Algorithm 8 provides a $(u/8, p/8)$-maxmin strategy.*

Proof. In order to show that the strategy obtained from Algorithm 8 is $(u/8, p/8)$-maxmin, we show that it achieves a payoff at least $u/8$ with probability at least $p/8$ against any pure strategy of player B. To this end, we consider two cases. Either the pure strategy of player B puts fewer than M' troops in the high-value battlefields, or puts no more than $M - M'$ troops in the low-value battlefields. We investigate both of the possibilities:

Fewer than M' troops in the high-value battlefields: Let M_h denote the amount of troops that player B puts in the high value battlefields. Note that $M' = N(|A|(1-p/2)+1)$ and in this case $M_h < M'$, which means $M_h < N(|A|(1-p/2)+1)$ and player B can put in at most $|A|(1-p/2)$ battlefields at least N troops. Therefore, when player A chooses a high-value battlefield uniformly at random, with probability at least $p/2$ player B puts less than N troops in that battlefield and player A wins it. As a result of playing this strategy player A wins at least a battlefield with probability at least $p/2$. Moreover, the payoff she achieves from winning any of these battlefields is at least $u/8$, so this guarantees a payoff of $u/8$ with probability at least $p/8 = (2/p)(1/4)$ since player A puts her troops in high-value battlefields with $1/4$ probability (by Line 16 of the algorithm).

No more than $M - M'$ troops in the low-value battlefields: We first prove that the summation of the weight of all the low-value battlefields is at least $u(M-M')/N$. We show that otherwise, player B can play in a way to prevent player A from obtaining a payoff of at least u with probability at least p. Recall that due to Line 14 of Algorithm 8, $M' = N(|A|(1-p/2)+1)$, therefore $2(|A|-\lfloor M'/N \rfloor)/|A| < p$ holds. It follows from Lemma 52 that player B can put M' troops in the high-value

battlefields to make sure player A wins no high-value battlefield with probability at least $1-p$. Therefore, there exists an strategy of player A that achieves a payoff of at least u from the low-value battlefields with non-zero probability. However, if

$$w_{b_1} + w_{b_2} + \ldots + w_{b_{|B|}} \geq u(M - M')/N$$

does not hold, the following strategy of player B can prevent player A from having that strategy : Let $W = \sum w_{b_i}$ be sum of the weights of the low-value battlefields. Put $(M - M')w_{b_i}/W$ troops in every battlefield b_i. Note that this requires no more than $M - M'$ troops since

$$\sum (M - M')w_{b_i}/W = (M - M')(\sum w_{b_i})/W = (M - M')W/W = M - M'.$$

In addition to this, in order for player A to win a battlefield b_i, she has to put more than $(M - M')w_{b_i}/W$ troops in that battlefield. Therefore, the ratio of the payoff over the number of necessary troop to win for each battlefield is at least $W/(M-M')$ and thus player A can obtain no more than $WN/(M - M')$ payoff. This implies $W \geq u(M - M')/N$ provided that there exists a (u, p)-maxmin strategy for player A.

Using this lower bound on the total size of the low-value battlefields, we also prove that $|\Delta| \geq \frac{8}{5}(M - M')/N$. Since by Line 6 of the algorithm, for any bundle of battlefields $\delta \in \Delta$, the total weight of the battlefields in δ is less than $\frac{5}{8}u$ $(= \frac{1}{2}u + \frac{1}{8}u)$,

$|\Delta|$ is bounded as follows:

$$|\Delta| \geq \frac{u(M-M')/N}{u5/8} \geq \frac{8}{5}(M-M')/N.$$

In Algorithm 8, with probability at least 3/4 player A plays on the low-value battlefields. She chooses a bundle δ of set Δ uniformly at random and distributes her money over all the battlefields in this bundle proportional to their weights. In this case if player B puts at most $\frac{3}{4}N$ troops in this bundle player A wins at least $\frac{1}{4}$ fraction of the overall weight of the battlefields in bundle δ which is at least $u/8$. Since $|\Delta| \geq \frac{8}{5}(M-M')/N$, and player B can put in at most $\frac{(M-M')}{3N/4}$ bundles at least $\frac{3}{4}N$ troops, there are at least

$$\frac{8(M-M')}{5N} - \frac{4(M-M')}{3N} \leq \frac{4(M-M')}{15N}$$

bundles that player B puts at most $\frac{3}{4}N$ troops in them, which is $\frac{1}{6}$ of all the bundles. As a result, in this case with probability at least $\frac{1}{8} = \frac{1}{6} \cdot \frac{3}{4}$ player A gains $\frac{u}{8}$ payoff since by line 16 of the algorithm she plays this strategy on the low-value battlefidls with probability at least $\frac{3}{4}$.

□

4.5 Auditing Game

In what follows we propose an approximation solution for the auditor. We first begin by showing an upper-bound on the highest protection that the auditor

can provide, and then proceed by proposing a strategy to obtain a fraction of that.

Note that if the hacker chooses a set of states with total value less than $\sum v_i - u$, the auditor is guaranteed to receive a payoff of at least u. Hence we assume throughout that any valid strategy of the hacker chooses a set of states that have a total value of more than $\sum v_i - u$.

We show an upper bound on the best that the auditor can do via a linear program. In this linear program, for every pure strategy x of the hacker, there is a variable $f_x \geq 0$ which is a real value corresponding to this strategy. We refer to f_x as the flow of strategy x. Moreover, for every state v_i, we have a constraint to ensure that the total flow of the strategies of the hacker that change the result of state v_i is bounded by 1. The objective of the program is to maximize the total flow of all strategies.

$$\begin{aligned} &\text{maximize:} && \sum_{x \in X} f_x && (4.5) \\ &\text{subject to:} && \sum_{s_i \ni X} f_x \leq 1 && \forall i \in [n] \end{aligned}$$

Let OPT be the optimal solution of LP 4.5. We show that no strategy of the auditor can achieve a utility of u with probability more than m/OPT. Of course, if $\text{OPT} \leq m$, this bound is meaningless. To this end, let f^* be an optimal solution of LP 4.5 and consider a mixed strategy of the hacker that plays every strategy x with probability f^*_x/OPT (notice that the probabilities sum up to 1 since $\text{OPT} = \sum f^*_x$).

Recall that the total flow of the strategies that hack every state is bounded by 1, and the hacker plays every strategy with probability f_x^*/OPT, thus the probability that any strategy of the hacker changes the outcome of any state is bounded by $1/\text{OPT}$. Moreover, every strategy of the auditor is to inspect at most m states, thus she catches the hacker with a probability of at most m/OPT. Hence, no strategy of the auditor can protect the election with a probability more than m/OPT.

Lemma 63. *There is no (u, p')-maxmin strategy for the auditor for $p' > m/\text{OPT}$, where OPT is the optimal solution of LP 4.5.*

Now, in order to find an approximation solution for the auditor, we take the dual of LP 4.5 to obtain the following linear program.

$$\begin{aligned} &\text{minimize:} && \sum_{i\in[n]} g_i && (4.6)\\ &\text{subject to:} && \sum_{s_i\in x} g_i \geq 1 \qquad \forall x \in X \end{aligned}$$

By the strong duality theorem, the optimal solutions of LP 4.6 and LP 4.5 are equal. Consider an optimal solution g^* of LP 4.6. Based on this solution, we construct a strategy for the auditor that protects the election with probability at least $(1-1/e)m/\text{OPT}$. Notice that in LP 4.6, for every state s_i, we have a variable g_i and the total sum of the variables is equal to OPT. Moreover, for every strategy x of the hacker, we have a constraint to ensure that the sum of the g_i^*'s corresponding to x is at least 1. Consider a probability distribution D over the states, such that for

every state s_i, $D_{s_i} = g_i^*/\text{OPT}$. Trivially, $\sum D_{s_i} = 1$ holds. Now, we define a strategy for the auditor and show its approximation factor is bounded by $1 - 1/e$. In this strategy, we draw m states from the probability distribution D and investigate the results of those states. Notice that some states might appear several times in our solution, but we can ignore repetitions and consider each just once.

Now, let us analyze this strategy. Fix a pure strategy x for the hacker. We know that the sum of $\{g_i^*\}$ for the states corresponding to strategy x is at least 1. Therefore, every time we draw a state according to the probability distribution D, one of the states of x is audited with probability at least $1/\text{OPT}$. We draw m different states, therefore, the probability that one of the states of x appears in the strategy of the auditor is at least

$$1 - (1 - 1/\text{OPT})^m \geq (1 - 1/e)m/\text{OPT}.$$

If $m \geq \text{OPT}$, then

$$\begin{aligned} 1 - (1 - 1/\text{OPT})^m &\geq 1 - (1 - 1/\text{OPT})^{\text{OPT}} \\ &\geq (1 - 1/e) \end{aligned}$$

and hence the proof is trivial. Otherwise,

$$1 - (1 - 1/\text{OPT})^m \geq (1 - 1/e)m/\text{OPT}$$

Thus, if the auditor follows this strategy, she can protect the election with

probability at least $(1-1/e)m/\text{OPT}$. Notice that solving the linear programs can be done in polynomial time via the ellipsoid method, and thus we can find this strategy in polynomial time. This result married with the upper-bound mentioned earlier gives us the following theorem.

Theorem 64. *For any given u and p, where a (u,p)-maxmin strategy is guaranteed to exist for the auditor, a polynomial time algorithm exists that finds a $(u,(1-1/e)p)$-maxmin approximation strategy.*

4.5.1 A Reduction From Generalized Blotto to Auditing Game

In a generalized version of Colonel Blotto which we call Threshold Blotto game, we have two players A and B, each with a given number of troops. Both of the players distribute their troops over k battlefields and the payoff of each battlefield i is determined based on a specific function h_i with respected to the troops of each side in battlefield i. If the total payoff of player A is more than a threshold τ then player A wins the game, otherwise player B is declared the winner.

Threshold Blotto game

Players: Colonels A and B

Settings: x and y specify the number of troops of the players. A set of k battlefields with payoff functions $\{h_1, h_2, \ldots, h_k\}$, where each function h_i admits two inputs α and β corresponding to the number of troops of the players in that battlefield and determines the payoff based on these values. A threshold τ denoting the minimum payoff for A to win the game.

Players' pure strategies: Distribute their troops over k battlefields. This can be represented with a vector of size k whose sum of indices is equal to the number of troops of that player.

Payoffs: A wins if $\sum h_i(\alpha_i, \beta_i) \geq \tau$ where α and β are vectors of size k for players strategies. If A wins, she receives a payoff of 1, otherwise a payoff of -1. Colonel B's payoff is the negation of colonel A's payoff and thus the game is zero-sum.

In what follows, we show a reduction from Threshold Blotto to the auditing game. Suppose we have an Auditing game with n states $s_1, s_2, \ldots, s_n$ and the hacker needs to flip the results of some states with a total number of electoral votes of at least t and the auditor audits the results of at most m states. Based on this, we construct an instance of the Threshold Blotto game with $k = n + 1$ battlefield. We associate colonel A to the hacker and colonel B to the auditor. We set the number of troops of colonel A equal to the total number of electoral votes of all states ($\sum v_i$) and the number of troops of Colonel B equal to m.

To make the reduction cleaner, we set the payoff function of each battlefield i (h_i) in such a way that the optimal strategies meet the following conditions:

- For every battlefield $1 \leq i \leq n$, player A either puts 0 or v_i troops (the number of electoral votes of state s_i) in the i'th battlefield.
- For every battlefield $1 \leq i \leq n$, player B either allocated 0 or 1 troop to that battlefield.
- The total number of troops of player A in the first n battlefields is at least t.

If all these conditions are met then, every time colonel A puts v_i troops in some battlefield $1 \leq i \leq n$, we can think of that as if the hacker hacks state s_i's results. Similarly, whenever colonel B puts a troop in a battlefield $1 \leq i \leq n$, we can interpret that as the auditor issuing a recount for state s_i. Therefore, we can define the payoff functions accordingly: For $1 \leq i \leq n$ we set

$$h_i(\alpha_i, \beta_i) = \begin{cases} -\infty, \text{ if } \beta_i \notin \{0,1\} \\ \infty, \quad \text{if } \alpha_i \notin \{0, v_i\} \\ 0 \qquad \text{if } \alpha_i = 0 \text{ or } \beta_i = 0 \\ 1 \qquad \text{if } \alpha_i > 0 \text{ and } \beta_i > 0 \end{cases}$$

where α_i and β_i denote the number of troops of colonels A and B in battlefield i respectively. This payoff function specifies the payoff of colonel B. The payoff of colonel A is the nagation of that. This guarantees that in an NE we always have $\alpha_i = \{0, v_i\}$ and $\beta_i = \{0, 1\}$ for all $1 \leq i \leq n$. Notice that colonel B gets a payoff

of 1 if she puts a troop in a battlefield and colonel A also puts v_i troops in that battlefield. To make sure the total number of electoral votes of the hacker is at least t, we set the following payoff function for the last battlefield:

$$h_k(\alpha_i, \beta_i) = \begin{cases} -\infty, \text{ if } \beta_i \neq 0 \\ \infty, \quad \text{if } \alpha_i > \sum v_i - t \\ 0 \qquad \text{otherwise} \end{cases}$$

Notice that if $\alpha_i \leq \sum v_i - t$, then the total number of troops of colonel A in the first n battlefield is at least t and thus the corresponding strategy of the hacker is valid. Now, colonel B gets a non-zero payoff in this game if and only if she puts some troops in a battlefield which also contains some troops of colonel A as well. Therefore, if we set $\tau = 1$ an NE of this game corresponds to an NE of the auditing game.

Theorem 65. *Threshold Blotto game is computationally harder than the auditing game.*

Algorithm 8: An algorithm to find a $(u/8, p/8)$-maxmin strategy of player A in continuous Blotto

$A = \{a_1, a_2, \ldots, a_{|A|}\} \leftarrow$ the set of battlefield with weight at least $u/8$
$B = \{b_1, b_2, \ldots, b_{|B|}\} \leftarrow$ the set of battlefield with weight less than $u/8$
$s \leftarrow 0$
$i \leftarrow 1$
for b in B **do**
    if $s \geq u/2$ **then**
        Add δ_i to Δ
        $i \leftarrow i + 1$
        $s \leftarrow 0$
    end if
    Add b to set δ_n.
    $s \leftarrow s +$ weight of battlefield b
end for
$M' \leftarrow N(|A|(1 - p/2) + 1)$
$coin \leftarrow 0$
With 3/4 probability set *coin* to 1.
if $coin = 0$ and $|A| = 0$ **then**
    $coin \leftarrow 1$
end if
if $coin = 1$ and ($|B| = 0$ or $M' > M$) **then**
    $coin \leftarrow 0$
end if
if $coin = 0$ **then**
    Choose a battlefield a uniformly at random from set A.
    Put N troops in battlefield a.
else
    Choose a set δ from Δ uniformly at random.
    for b in δ **do**
        Put N/w_b troops in battlefield b.
    end for
end if

Chapter 5: Solutions with Small Profiles

Throughout the chapter, for any integer n, we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. We denote the vectors by bold fonts and for every vector $\mathbf{v}$, use v_i to denote its ith entry.

The Colonel Blotto game. The Colonel Blotto game is played between two players which we refer to as player 1 and player 2. Any instance of the game can be represented by a tuple $\mathcal{B}(n, m, \mathbf{w})$ where n and m respectively denote the number of *troops* of player 1 and player 2, and $\mathbf{w} = (w_1, \ldots, w_k)$ is a vector of length k of positive integers denoting the *weight* of k *battlefields* on which the game is being played.

A pure strategy of each of the players is a partitioning of his troops over the battlefields. That is, any pure strategy of player 1 (resp. player 2) can be represented by a vector $\mathbf{x} = (x_1, \ldots, x_k)$ of length k of non-negative numbers where $\sum_{i\in[k]} x_i \leq n$ (resp. $\sum_{i\in[k]} x_i \leq m$). In the *discrete* variant of the game, the number of troops that are assigned to each of the battlefields must be non-negative integers. In contrast, in the *continuous* variant, any assignment with non-negative real values is considered valid. Throughout the chapter, we denote respectively by $\mathcal{S}_1$ and $\mathcal{S}_2$ the set of all valid pure strategies of player 1 and player 2.

Let $\mathbf{x}$ and $\mathbf{y}$ be the pure strategies that are played respectively by player 1 and player 2. Player 1 wins battlefield i if $x_i > y_i$ and loses it otherwise. The winner of battlefield i gets a partial utility of w_i on that battlefield, and the overall utility of each player is the sum of his partial utilities on all the battlefields. More precisely, the utilities of players 1 and 2, which we respectively denote by $\mathsf{u}_1(\mathbf{x}, \mathbf{y})$ and $\mathsf{u}_2(\mathbf{x}, \mathbf{y})$, are as follows:

$$\mathsf{u}_1(\mathbf{x}, \mathbf{y}) = \sum_{i \in [k]: x_i > y_i} w_i, \qquad \mathsf{u}_2(\mathbf{x}, \mathbf{y}) = \sum_{i \in [k]: x_i \leq y_i} w_i.$$

Note that in the definition above, we break the ties in favor of player 2, i.e., when both players put the same number of troops on a battlefield, we assume the winner is player 2.

Also, we define the *uniform* Colonel Blotto game to be a special case of Colonel Blotto in which all of the battlefields have the same weights, i.e., $w_1 = w_2 = \ldots = w_k = 1$.

The objective. The guaranteed expected utility of a (possibly mixed) strategy $\mathbf{x}$ of player 1 is u, if against any (possibly mixed) strategy $\mathbf{y}$ of player 2, we have $\mathbb{E}_{\mathbf{x}' \sim \mathbf{x}, \mathbf{y}' \sim \mathbf{y}}[\mathsf{u}_1(\mathbf{x}', \mathbf{y}')] \geq u$. A strategy is *maximin* if it maximizes this guaranteed expected utility.

In this chapter, we are interested in bounded *support* strategies. The support of a mixed strategy is the set of pure strategies to which it assigns a non-zero probability. We call a strategy c-strategy if its support has size at most c. A maximin c-strategy is a c-strategy that has the highest guaranteed expected utility among all

c-strategies. Observe that we do not restrict the adversary to play a c-strategy here. In fact, given the mixed strategy of a player, it is well-known that the best response of the opponent can be assumed to be a pure strategy w.l.o.g. This means that an opponent that can respond by only one pure strategy is as powerful as an opponent that can play mixed strategies as far as maximin strategies are concerned.

Another standard objective for many natural applications of the Colonel Blotto game is to maximize the probability of obtaining a utility of at least u. This has been captured in the literature by the notion of (u, p)-maximin strategies.

Definition 66. *For any two-player game, a (possibly mixed) strategy* $\mathbf{x}$ *of player 1 is called a* (u, p)-*maximin strategy, if for any (possibly mixed) strategy* $\mathbf{y}$ *of player 2,*

$$\Pr_{\mathbf{x}' \sim \mathbf{x}, \mathbf{y}' \sim \mathbf{y}} [\mathsf{u}(\mathbf{x}', \mathbf{y}') \geq u] \geq p.$$

In such scenarios, for a given minimum utility u, our goal is to compute (or approximate) a (u, p)-maximin with maximum possible p. Similar to the expected case, we are interested in bounded support strategies. That is, given u and c, our goal is to compute (or approximate) a (u, p)-maximin c-strategy with maximum possible p. Again, we do not restrict the adversary's strategy.

Throughout this section, we discuss the optimal strategies of the Colonel Blotto game when the support size is small. That is, when player 1 can only randomize over at most c pure strategies and wishes to maximize his utility. Our main interest is in finding (almost) optimal (u, p)-maximin strategies as discussed above. Nonetheless, we show in Section 5.5 that our results carry over to the conventional definition of

the maximin strategies and can be used when the goal is to maximize the expected utility.

When randomization is not allowed, any (u, p)-maximin 1-strategy with $p > 0$ is also $(u, 1)$-maximin. Thus, the problem boils down to finding a pure strategy with the maximum guaranteed payoff. However, when randomization over two pure strategies is allowed, we may play two pure strategies with different probabilities q and $1-q$ and thus the problem of finding a (u, p)-maximin 2-strategy with $p \neq 1$ does not necessarily reduce to the case where the goal is to find a single pure strategy. As an example, when $n = m = 2$ and we have two battlefields with equal weights ($w_1 = w_2 = 1$), a $(1, 1/2)$-maximin 2-strategy can be obtained by selecting a battlefield at random and placing two troops on the selected battlefield. However, in this example (or in any example with $m = n$), no 2-strategy is $(1, p)$-maximin for $p > 1/2$ since the opponent can just select our higher-probability strategy and copy it, ensuring we get utility 0 with probability at least $1/2$. More generally, for $c = 2$, even if $n \leq m$, a simple observation shows that the only interesting cases are when $p = 1$ (which reduces to the $c = 1$ case) or when $p = 1/2$. The idea is that when $p > 1/2$ holds, existence of a (u, p)-maximin 2-strategy for an arbitrary u implies that of a $(u, 1)$-maximin 2-strategy. Similarly, if a 2-strategy is (u, p)-maximin for some u and $0 < p < 1/2$, one can modify the same strategy to make it $(u, 1/2)$-maximin. Therefore, for $c = 2$, the problem is (computationally) challenging only when a $(u, 1/2)$-maximin 2-strategy is desired. Thus, for $c = 2$, the problem essentially reduces to finding two pure strategies $\mathbf{x}, \mathbf{x}'$ such that no strategy of the opponent can prevent *both* $\mathbf{x}$ and $\mathbf{x}'$ from getting a utility of at least u. A similar, but more in-depth analysis gives us

the same structure for $c = 3$. That is, in this case, we may look for a $(u, 1/3)$-maximin or a $(u, 2/3)$-maximin 3-strategy. This implies that in an optimal solution, we look for three strategies $\mathbf{x}, \mathbf{x}', \mathbf{x}''$ such that no strategy of the opponent prevents two of or all of (depending on whether $p = 1/3$ or $p = 2/3$) $\mathbf{x}, \mathbf{x}'$ and $\mathbf{x}''$ from getting a utility of at least u.

It is surprising to see that this structure breaks when considering more than three pure strategies ($c \geq 4$). For instance, consider an instance of the Colonel Blotto game with 4 battlefields ($k = 4$) in which the players have 4 and 6 troops, respectively ($n = 4$, $m = 6$). Let the weights of the first 3 battlefields be 5 and the weight of the last battlefield be 10. In this example, the goal of player 1 is to obtain a utility of at least 10 with the highest probability. One can verify with an exhaustive search that if player 1 were to randomize over 4 pure strategies with equal probability, he could guarantee a utility of at least 10 with probability no more than $1/4$.[1] However, we present in Table 5.1, a $(10, 2/5)$-maximin 4-strategy for player 1 that plays 4 pure strategies with non-uniform probabilities.

	Battlefield 1 $w_1 = 5$	**Battlefield 2** $w_2 = 5$	**Battlefield 3** $w_3 = 5$	**Battlefield 4** $w_4 = 10$
Strategy 1, played with probability $2/5$.	0	0	0	4
Strategy 2, played with probability $1/5$.	1	1	2	0
Strategy 3, played with probability $1/5$.	1	2	1	0
Strategy 4, played with probability $1/5$.	2	1	1	0

Table 5.1: A $(10, 2/5)$-maximin 4-strategy for player 1 on instance $\mathcal{B}(4, 6, (5, 5, 5, 10))$.

In order to design an algorithm for finding (u, p)-maximin strategies that does not lose on p, it is essential to understand how player 1 randomizes over his strategies

[1] We have verified this claim with a computer program.

in an optimal solution. We begin in Section 5.2 by showing that when randomization is allowed only on a small number of pure strategies, the number of different ways to distribute the probability over the pure strategies in an optimal solution is limited. That is, one can list a number of probability distributions and be sure that at least one of such probability distributions leads to an optimal solution. This structural property is general and applies to any game so long as (u, p)-maximin strategies are concerned.

Theorem 77 (restated informally). *When randomization is only allowed on a constant number of pure strategies, the number of possible probability distributions for an optimal (u, p)-maximin strategies is limited by a constant.*

Theorem 77 is proven via a combinatorial analysis of the optimal solutions. On one hand, we leverage the optimality of the solution to argue that p cannot be improved. On the other hand, we use the maximality of p to derive relations between the probabilities that the pure strategies are played with. Finally, we use these relations to narrow down the probabilities a to a small set.

Indeed, a consequence of Theorem 77 is that when we fix a utility u and wish to find a (u, p)-maximin c-strategy with highest p, p can only take a constant number of values. This observation makes the problem substantially easier as we can iterate over all possible probability profiles and solve the problem separately for each profile. Thus, from now on, we assume that we fix a probability distribution over the pure strategies, and the goal is to select c strategies to be played according to the fixed probabilities. We assume that (i) u and p are given in the input, (ii)

we are guaranteed that a (u, p)-maximin c-strategy exists, and (iii) the goal is to compute/approximate such a strategy.

Continuous Colonel Blotto. For the continuous case, we can represent each pure strategy with a vector of size k of real values indicating the number of troops that is placed on each battlefield. Thus, a c-strategy can be represented by c vectors of length k (given that we have fixed the probability distribution over the pure strategies). A simple observation (also pointed out in Chapter 4) is that when the goal is to find a pure maximin strategy, the solution is essentially a convex set with respect to the representations. That is, if $\mathbf{x}^1$ and $\mathbf{x}^2$ are both (u, p)-maximin, any strategy whose representation is a convex combination of the representations of $\mathbf{x}^1$ and $\mathbf{x}^2$ is also (u, p)-maximin. The pure maximin problem coincides with our setting when $c = 1$. However, it is surprising to see that for $c > 2$, the solution may not be convex, even though we use the same approach to represent the strategies.

As an example, imagine we have two battlefields with equal weights (say 1) and $n = m = 2$. Indeed a $(1, 1/2)$-maximin strategy can be obtained for player 1 by randomizing over $(0, 2)$ and $(2, 0)$ uniformly. Similarly, randomizing over $(2, 0)$ and $(0, 2)$ (the order is changed) gives us the same guarantee. However, a convex combination of the two strategies plays $(1, 1)$ deterministically and loses both battlefields against the strategy $(1, 1)$ of player 2. The situation may be even worse as one can construct a delicate instance whose solution set is the union of up to $2^{\Omega(k)}$ convex regions no two of which make a convex set when merged.

A key observation that enables us to compute/approximate the solution is

the *partial convexity* of the solution. That is, we show that although the solution set is not necessarily convex, one can identify regions of the space where for each region, the solution is convex. To be more precise, denote by $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^c$ the representations of the strategies. In this representation, x^i_j denotes the number of troops that i'th pure strategy places on the j'th battlefield. This gives us a ck dimensional problem space $[0, n]^{ck}$. Divide this space by $k\binom{c}{2}$ hyperplanes each formulated as $x^i_j = x^{i'}_j$ for some $i, i' \in [c]$ and $j \in [k]$. Partial convexity implies that the solution space in each region is convex, and as a result, gives us an exponential time solution for the problem in the continuous setting. Roughly speaking, since the solution is convex in each region, we can iterate over all regions and solve the problem separately using a linear program for each region. However, we have exponentially many regions and therefore the running time of this approach is exponentially large.

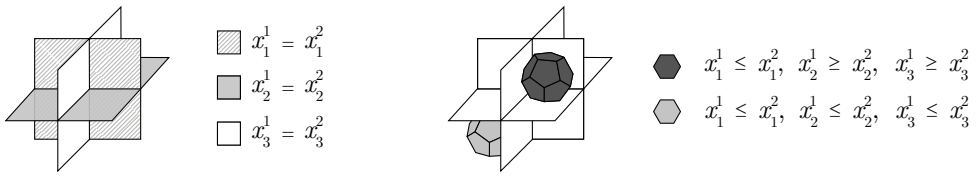

Figure 5.1: An example of how the solution space is decomposed for the case when $k = 3$ and $c = 2$. The figure on the left illustrates 3 hyperplanes $x^1_j = x^2_j$. These hyperplanes divide the solution space into 8 regions and for all the points in each region, as illustrated in the figure on right, x^1_j and x^2_j for $j \in [3]$ compare in the same way.

The above algorithm can be modified to run in polynomial time in the uniform setting. The high-level idea is that when the battlefields have equal weights, there is a strong symmetry between the solutions of the regions. Based on this, we show that in the uniform setting, one only needs to search a polynomial number of regions for a solution. This idea along with the partial convexity gives us a polynomial time

solution for the uniform setting.

Theorem 71 (restated informally – proven in Section 5.1.1 for $c = 2$ and extended in Section 5.3.1 to $c > 2$). *There exists a polynomial time solution for finding a* (u, p)-*maximin* c-*strategy for Colonel Blotto in the continuous case when all the battlefields have the same weight.*

Indeed, the uniform setting is a very special case since we are essentially indifferent to the battlefields. When we incorporate the weights of the battlefields, we no longer expect the solutions to be symmetric with respect to the battlefields. However, one may still observe a weak notion of symmetry between the regions. Recall that we denote the weights of the battlefields with $w_1, w_2, \ldots, w_k$. Let us lose a factor of $1 + \epsilon$ in the utility and assume w.l.o.g. that each weight w_i is equal to $(1+\epsilon)^j$ for some integer $j \geq 0$. Assuming that the maximum weight is polynomially bounded (we only make this assumption for the sake of simplicity and our solution does not depend on this constraint), the number of different battlefield weights is logarithmically small. Thus, we expect many battlefield to have equal weights which as a result makes the solution regions more symmetric. However, this idea alone gives us a quasi-polynomial time algorithm for searching the regions as we may have a logarithmic number of different battlefield weights. To reduce the running time to polynomial, we need to further prune the regions of the solution to polynomially many. Indeed, we show that it suffices to search over a polynomial number of regions if we allow an approximate solution. Via this observation, we can design a polynomial time algorithm that approximates the solution within a factor $1 + \epsilon$.

This settles the problem for the continuous setting.

Theorem 84 (restated informally – proven in Section 5.1.1 for $c = 2$ and extended in Section 5.3.1 to $c > 2$). *(u, p)-maximin c-strategy strategies of the Colonel Blotto admit a PTAS in the continuous setting.*

Discrete Colonel Blotto. For the discrete setting, we take a rather different approach. The main reason is that in this setting, even if we are guaranteed that the solution is convex, we cannot use LP's to compute/approximate a solution. In Chapter 4 we give a 2 approximation algorithm that finds a pure maximin strategy for the discrete setting. Indeed, this solution can be used to get a 2 approximate solution for the case of $c = 1$. We both extend their algorithm to work for $c \geq 2$ and also devise a *heavy-light decomposition* to improve the approximation factor to $1 + \epsilon$. Both our extension and decomposition techniques are novel.

We introduce the notion of a *weak adversary*. Roughly speaking, we define a relaxed best response for player 2 that does not maximize the utility of player 2, but instead, approximately maximizes his utility. We call a player that plays a relaxed best response, a weak adversary. By proposing a greedy algorithm for the weak adversary, we show that the payoff of the weak adversary and the actual adversary differ by the value of at most one battlefield. That is, if the weights of the battlefields are bounded by $\mathsf{w}_{\mathsf{max}}$, then the difference between the utility of an adversary and that of a weak adversary is always bounded by $\mathsf{w}_{\mathsf{max}}$. Next, we show that a dynamic program can find a pure strategy of player 1 that performs best against a relaxed adversary and turn this algorithm into a 2 approximate solution

for the problem, by considering two cases $2\mathsf{w}_{\mathsf{max}} \geq u$ and $2\mathsf{w}_{\mathsf{max}} < u$.

Indeed losing an additive error of $\mathsf{w}_{\mathsf{max}}$ may hurt the approximation factor a lot when the desired utility u is not much more than $\mathsf{w}_{\mathsf{max}}$. Thus, in order to improve the approximation factor, one needs to design a separate algorithm for the heavy battlefields. To this end, we introduce our heavy-light decomposition. We define a threshold $\tau \approx \epsilon u$ and call a battlefield i heavy if $w_i > \tau$ and light otherwise. In addition to this, we assume w.l.o.g. that $\mathsf{w}_{\mathsf{max}} \leq u$ since otherwise one can set a cap of u on the weights without changing the solution. Therefore, the maximum weight and the minimum weight of heavy battlefields are within a multiplicative factor of $1/\epsilon$. Next, by incurring an additional $1+\epsilon$ multiplicative factor to the approximation guarantee, we round down the weight of each battlefield to the nearest $(1+\epsilon)^i$. We show that this leaves us with a constant number of different weights for the heavy battlefields. Next, we state that since the number of different weights for the heavy battlefields is constant, the total number of (reasonable) pure strategies of player 1 over these battlefields can be reduced down to a polynomial. Indeed this also holds for player 2, but for the sake of our solution, we should further bound the number of responses of player 2 over these battlefields. We show in fact, that the number of (reasonable) responses of player 2 over the heavy battlefields against a strategy of player 1 is bounded by a constant! For light battlefields, we use the idea of a weak adversary. However, in order to find a solution that considers both heavy and light battlefields, we need to define multiple weak adversaries each with regard to a response of player 2 on the heavy battlefields.

Let us clarify the challenge of mixing the two solutions via an example. Sup-

pose we have 4 battlefields with weights $10, 8, 7$ and 5 and the players' troops are as follows: $n = 5$ and $m = 2$. One can verify that in this case, the following pure strategy of player 1 guarantees a payoff of 15 for him.

	Battlefield 1 $w_1 = 10$	**Battlefield 2** $w_2 = 8$	**Battlefield 3** $w_3 = 7$	**Battlefield 4** $w_4 = 5$
A $(15, 1)$-maximin 1-strategy for player 1.	2	2	1	0

Table 5.2: A $(15, 1)$-maximin 1-strategy for player 1 on instance $\mathcal{B}(5, 2, (10, 8, 7, 5))$.

In fact, 15 is the highest utility player 1 can get with a single pure strategy as no other pure strategy of player 1 can guarantee a payoff more than 15 for him. Now, assume that we select the first two battlefields with weights 10 and 8 as heavy battlefields and the rest of the battlefields as the light ones. One may think that by taking a maximin approach for the heavy battlefields and solving the problem separately for the light battlefields, we can obtain a correct solution. The above example shows that this is not the case. We show in what follows, that the maximin approach reports a payoff of 17 for player 1 which is more than the actual solution. Fix the strategy of player 1 on the heavy battlefields to be placing 2 troops on battlefield 1 and 1 troop on battlefield 2. As such, the only reasonable responses of player 2 on these battlefields are as shown in the following table.

	Battlefield 1 $w_1 = 10$	**Battlefield 2** $w_2 = 8$	troops left for player 2
response 1	2	0	0
response 2	0	1	1
response 3	0	0	2

Table 5.3: The responses of player 2 on the heavy battlefields.

Response 3 already gives player 1 a payoff of 18 which is more than 17. Also, response 1 of player 2 leaves him with no troops for the light battlefields and thus he loses both light battlefields against strategy $(1, 1)$ of player 1 on the light battlefields. Therefore, this gives player 1 a payoff of 20. Also, if player 2 plays response 2 on the heavy battlefields, player 1 can win the light battlefield w_3 by putting two troops on it. Indeed player 2 has only one troop left and there is no way for him to win this battlefield. Thus, in this case, the payoff of player 1 would be 17. Since we take the maximum solution over all strategies of player 1 for the heavy battlefields, our final utility would be at least 17.

What the above analysis shows is that, if we take a maximin approach on the heavy battlefields and then solve (or approximate) the problem for the light battlefields, we may incorrectly report a higher payoff for player 1. Roughly speaking, this error happens since in this approach, we allow player 1 to have different actions over the light battlefields against different responses of player 2. To resolve this issue, we design a dynamic program that takes into account all responses of player 2 simultaneously. Indeed, to make sure the program can be solved in polynomial time, we need to narrow down the number of responses of player 2 to a constant. Our heavy-light decomposition along with our structural properties of the optimal strategies enables us to reach this goal. This gives us a non-trivial dynamic program that can approximate the solution within a factor $1 + \epsilon$ for the case of a single pure strategy.

Theorem 85 (restated informally). *The problem of finding a* $(u, 1)$*-*maximin *strategy*

for player 1 in discrete Colonel Blotto admits a PTAS.

To extend the result to the case of $c \geq 2$, we need to design a weak adversary that plays a relaxed best response against more than 1 strategy of player 1. For $c = 1$, the greedy algorithm follows from the well-known greedy solution of knapsack. However, when $c \geq 2$ the best-response problem does not necessarily reduce to knapsack and therefore our greedy solution is much more intricate. Roughly speaking, we design a non-trivial procedure for player 2 that gets c thresholds as input, and based on these thresholds, decides about the response for each battlefield locally. This local decision making is a key property that we later exploit in our dynamic program to find an optimal strategy against a weak adversary. This in addition to the heavy-light decomposition technique gives us a PTAS for (u, p)-maximin c-strategy strategies of Colonel Blotto in the discrete setting.

Theorem 108 (restated informally). *The problem of finding a (u, p)-maximin c-strategy for player 1 in discrete Colonel Blotto admits a PTAS.*

Further results. We show in Section 5.5, that our techniques also imply PTASs for the Colonel Blotto game when instead of a (u, p)-maximin c-strategy we wish to find a maximin c-strategy. For the continuous case, similar to the case of (u, p)-maximin strategies, we divide the solution space into polynomially many convex subregions and prove that among them a $(1 + \epsilon)$-approximate solution is guaranteed to exist. The main difference with the case of (u, p)-maximin strategies is in the LP formulation of the problem, but the general approach is essentially the same. For the discrete variant of Colonel Blotto, we also follow a similar approach as in the case

of (u, p)-maximin strategies. In more details, we partition the battlefields into heavy and light subsets and define a weaker adversary that is adapted to approximately best respond against maximin strategies. We find it surprising and possibly of independent interest that essentially the same approach (though with minor changes) can be applied to these two variants of Colonel Blotto. Prior algorithms proposed for these two variants were fundamentally different [1, 3, 54].

Theorems 120 and 124 (restated informally). *The problem of finding a maximin c-strategy for player 1 in both discrete and continuous variants of Colonel Blotto admits a PTAS.*

Finally, recall that motivation for approximate algorithms comes from intractability. In view of all the recent sophisticated algorithmic approaches to solving, approximately, various special cases of the Colonel Blotto game, it is worth asking, what is the computational complexity of the full fledged problem of computing a maximin strategy of the Colonel Blotto game? (Notice that, since the full strategy is too long to return, we should formulate the problem in terms of something succinct, for example *one* component of the maximin.) In Section 5.6 we present the first complexity results in this area, establishing that an interesting variant of the problem that we call General Colonel Blotto — roughly, the utility is a general function of the two allocations, instead of the probability of winning more than a certain goal of total battlefield weight — is *complete for exponential time.* The precise complexity of the two versions of the original game (computing a maximin of the probability of winning a majority, or of the expectation of the total weights of

battlefields won) is left as an open question here. We conjecture that both problems are also exponential time-complete.

Theorems 125. General Colonel Blotto *is exponential time-complete.*

We formalize our algorithm for the continuous case in the remainder of section. In Section 5.2, we discuss possible probability distributions over the pure strategies in an optimal (u, p)-maximin c-strategy. In Section 5.4, we show that the discrete variant of the game also admits a PTAS. We further show how it is possible to adapt these results to the case of maximin strategies in Section 5.5. Finally, in Section 5.6, we describe our complexity results.

5.1 Continuous Colonel Blotto

In this section we consider the continuous variant of Colonel Blotto. We start with the case where our goal is to find a (u, p)-maximin 1-strategy and show how we can generalize it to 2-strategies and, further, to any bounded number of strategies.

For the particular case of $c = 1$, our goal is to find a single pure strategy that is (u, p)-maximin. Indeed, since we are playing a single pure strategy with no randomization, the probability p must be 1. One can think of the solution of this case as a vector of non-negative real values that sum up to n and formulate the

problem in the following way.

$$\begin{array}{lll} \text{find} & \mathbf{x} & \\ \text{subject to} & x_i \geq 0 & \forall i \in [k] \\ & \sum x_i \leq n & \\ & \mathsf{u}_1(\mathbf{x}, \mathbf{y}) \geq u & \forall \mathbf{y} \in \mathcal{S}_2 \end{array} \tag{5.1}$$

To analyze Program 5.1, we need to better understand the payoff constraints. To this end, we state another interpretation of the payoff constraints in Observation 67.

Observation 67 ([3])**.** *A pure strategy* $\mathbf{x}$ *of player 1 guarantees a payoff of at least* u *against any pure strategy of player 2 if and only if* $\sum_{i \in S} x_i > m$ *for any set* S *of battlefields with* $\sum_{i \notin S} w_i < u$*.*

Proof. ($\Rightarrow$): Suppose that a strategy $\mathbf{x}$ does not get a payoff of u against a strategy $\mathbf{y}$ of player 2. This means that there exists a set S of battlefields that player $\mathbf{x}$ loses. The total payoff of $\mathbf{x}$ is below u, and therefore $\sum_{i \notin S} w_i < u$. In addition to this $\sum_{i \in S} x_i \leq m$ holds since player 2 needs to match player 1's troops in all battlefields of S.

($\Leftarrow$): It is trivial to show that if there exists such a violating set S, then $\mathbf{x}$ cannot promise a payoff of u against any strategy of player 2. The reason is that since $\sum_{i \in S} x_i \leq m$ player 2 can match the troops of player 1 in set S and that suffices to prevent player 1 from winning a payoff of u. □

Via Observation 67, we can turn Program 5.1 into LP 5.2 where we have a constraint for every possible subset S of battlefields with $\sum_{i \notin S} w_i < u$. Although

the number of these subsets can be exponentially large, we show in Chapter 4 that one can find a violating constraint of LP 5.2 in polynomial time and thus find a feasible solution using the ellipsoid method.

$$\begin{aligned} \text{find} \quad & \mathbf{x} \\ \text{subject to} \quad & x_i \geq 0 \qquad \forall i \in [k] \\ & \textstyle\sum x_i \leq n \\ & \textstyle\sum_{i \in S} x_i > m \quad \text{for every subset } S \text{ of battlefields with } \sum_{i \notin S} w_i < u. \end{aligned} \tag{5.2}$$

The key idea that enables us to solve this variant is that we are playing only one pure strategy. Even the case of having two pure strategies in the support is much more challenging. To illustrate the challenges and ideas, we next focus on how to obtain a 2-strategy and later generalize it to c-strategies.

5.1.1 The Case of 2-Strategies

Recall by Observation 78 of Section 5.2 that if a 2-strategy is (u, p)-maximin for some $p > 1/2$, then there also exists a pure $(u, 1)$-maximin strategy that, by aforementioned techniques, can be found in polynomial time. It was further shown in Observation 79 that if a 2-strategy is (u, p)-maximin for some $p < 1/2$, we can simply play the two strategies in its support with equal probability $1/2$ to obtain a better $(u, 1/2)$-maximin strategy. Combining these two observations, we can assume w.l.o.g., that in the case of $c = 2$, we wish to find two pure strategies $\mathbf{x}$ and $\mathbf{x}'$,

and play them with equal probabilities such that any strategy of player 2 gives us a payoff of at least u for at least one of $\mathbf{x}$ or $\mathbf{x}'$. A mathematical formulation of the problem is given below.

$$\begin{array}{lll} \text{find} & \mathbf{x} \text{ and } \mathbf{x}' & \\ \text{subject to} & x_i \geq 0 \text{ and } x'_i \geq 0 & \forall i \in [k] \\ & \sum x_i \leq n & \\ & \sum x'_i \leq n & \\ & \text{either } \mathsf{u}_1(\mathbf{x}, \mathbf{y}) \geq u \text{ or } \mathsf{u}_1(\mathbf{x}', \mathbf{y}) \geq u & \forall \mathbf{y} \in \mathcal{S}_2 \end{array} \tag{5.3}$$

Observe that the fourth constraint of the above program is not linear. In the following, we show that even the polytope that is described by this program is essentially nonconvex.

Observation 68. *Program 5.3 is not convex.*

Proof. Suppose $n = m = 2$, $w_1 = w_2 = 1$ and the goal is to find a $(1, 1/2)$-maximin strategy by randomizing over two pure strategies. A possible solution is to play $\mathbf{x} = (2, 0)$ with probability $1/2$ and play $\mathbf{x}' = (0, 2)$ with probability $1/2$ which guarantees a payoff of 1 with probability $1/2$. An alternative way to achieve this goal is to set $\mathbf{x} = (2, 0)$ and $\mathbf{x}' = (0, 2)$ which is the same strategy except that $\mathbf{x}$ and $\mathbf{x}'$ are exchanged. However, the linear combination of the two strategies results in $\mathbf{x} = (1, 1)$ and $\mathbf{x}' = (1, 1)$ which always loses both battlefields against $\mathbf{y} = (1, 1)$. □

A more careful analysis shows that the feasible region of Program 5.3 may be the union of up to 2^k convex polytopes which makes it particularly difficult to find a

desired solution. In what follows, we present algorithms to overcome this challenge for both the uniform and nonuniform settings.

5.1.1.1 Uniform Setting

Recall that in order to find a pure $(u, 1)$-maximin strategy, we proved the linearity of Program 5.1 by characterizing the optimal solution. Similar to that, we show necessary and sufficient conditions for the solution of Program 5.3. To this end, we define *critical tuples* as follows.

Definition 69. *Let L_1, L_2, and L_{12} be three disjoint subsets of battlefields. We call the tuple $\langle L_1, L_2, L_{12}\rangle$ a critical tuple, if critical any of $L_1 \cup L_{12}$ or $L_2 \cup L_{12}$ suffices to prevent player 1 from getting a payoff of u. In other words, $\langle L_1, L_2, L_{12}\rangle$ is a critical tuple, if and only if*

$$\sum_{i \notin L_1 \cup L_{12}} w_i < u, \qquad \text{and,} \qquad \sum_{i \notin L_2 \cup L_{12}} w_i < u.$$

Via this definition, we can now describe the feasible solutions of Program 5.3 as follows.

Observation 70. *Two pure strategies $\mathbf{x}$ and $\mathbf{x}'$ of player 1 meet the constraints of Program 5.3 if and only if for any critical tuple $\langle L_1, L_2, L_{12}\rangle$ we have*

$$\sum_{i \in L_1} x_i + \sum_{i \in L_2} x'_i + \sum_{i \in L_{12}} \max\{x_i, x'_i\} > m.$$

Proof. The proof is similar to that of Observation 67. Note that $\mathbf{x}$ and $\mathbf{x}'$ violate a

payoff constraint of Program 5.3 if they both get a payoff less than u against a pure strategy $\mathbf{y}$ of player 2. In this case we define three sets L_1, L_2, and L_{12} as

$$L_1 = \{i : x_i \leq y_i \text{ and } x'_i > y_i\}, \quad L_2 = \{i : x_i > y_i \text{ and } x'_i \leq y_i\}, \quad L_{12} = \{i : x_i \leq y_i \text{ and } x'_i \leq y_i\}.$$

Observe that L_1, L_2, and L_{12} make a critical tuple since both $\mathbf{x}$ and $\mathbf{x}'$ get a payoff less than u against y. Since L_1, L_2, and L_{12} are disjoint and $\sum y_i = m$ we have $\sum_{i \in L_1} x_i + \sum_{i \in L_2} x'_i + \sum_{i \in L_{12}} \max\{x_i, x'_i\} \leq m$. A similar argument implies that if this condition does not hold for any critical tuple, then $\mathbf{x}$ and $\mathbf{x}'$ meet the conditions of Program 5.3. □

Based on Observation 70 we rewrite Program 5.3 in the following way.

$$\begin{array}{lll}
\text{find} & \mathbf{x} \text{ and } \mathbf{x}' & \\
\text{subject to} & x_i \geq 0 \text{ and } x'_i \geq 0 & \forall i \in [k] \\
& \sum x_i \leq n & \\
& \sum x'_i \leq n & \\
& z_i = \max\{x_i, x'_i\} & \forall i \in [k] \\
& \sum_{i \in L_1} x_i + \sum_{i \in L_2} x'_i + \sum_{i \in L_{12}} z_i > m & \text{for every critical tuple } \langle L_1, L_2, L_{12} \rangle
\end{array} \tag{5.4}$$

Indeed Program 5.4 is not convex since $z_i = \max\{x_i, x'_i\}$ is not a linear constraint. The naive approach to get around this issue is to consider 2^k possibilities for the assignment of z_i's. More precisely, if we knew in an optimal strategy for which i's we have $x_i > x'_i$ and for which $x_i \leq x'_i$, we could turn Program 5.4 into a

linear program by replacing each z_i with either x_i or x'_i. This observation gives us an exponential time solution to find a (u,p)-maximin strategy by trying all 2^k combinations. However, for the uniform case, we can further improve the running time to a polynomial. The overall idea is that when we are indifferent between the battlefields, we do not necessarily need to know for which subset of battlefields $\mathbf{x}$ puts more troops that $\mathbf{x}'$. It suffices to be given the count!

In the uniform setting, let $\hat{\mathbf{x}}$ and $\hat{\mathbf{x}}'$ be the actual solution. Count the number of battlefields on which $\hat{\mathbf{x}}$ puts more troops than $\hat{\mathbf{x}}'$ and call this number α. Therefore, on $k-\alpha$ battlefields, $\mathbf{x}'$ puts at least as many troops as $\mathbf{x}$. Since the battlefields are identical, we can rearrange the order of the battlefields to make sure $\mathbf{x}$ puts more troops than $\mathbf{x}'$ in the first α battlefields. If α is given to us, we can formulate the problem as follows.

$$
\begin{array}{lll}
\text{find} & \mathbf{x} \text{ and } \mathbf{x}' & \\
\text{subject to} & x_i \geq 0 \text{ and } x'_i \geq 0 & \forall i \in [k] \\
& \sum x_i \leq n & \\
& \sum x'_i \leq n & \\
& x_i \geq x'_i, z_i = x_i & \forall i : 1 \leq i \leq \alpha \\
& x_i \leq x'_i, z_i = x'_i & \forall i : \alpha < i \leq k \\
& \sum_{i \in L_1} x_i + \sum_{i \in L_2} x'_i + \sum_{i \in L_{12}} z_i > m & \text{for every critical tuple } L_1, L_2, L_{12}
\end{array}
\tag{5.5}
$$

Program 5.5 is clearly an LP. We show in Theorem 71 that LP 5.5 can indeed

be solved in polynomial time. This settles the problem when α is given. Note that α takes an integer value between 0 and k and thus we can iterate over all possibilities and solve the problem in polytime.

Theorem 71. *Given that an instance of continuous Colonel Blotto in the uniform setting admits a* $(u, 1/2)$-*maximin* 2-*strategy for player 1, there exists an algorithm to find one such solution in polynomial time.*

Proof. As discussed earlier, the solution boils down to solving LP 5.5. Here we show that LP 5.5 can be solved in polynomial time using the ellipsoid method. For that, we need a separating oracle that for any given assignment to the variables decides in polynomial time whether any constraint is violated and if so, reports one. LP 5.5 has polynomially many constraints, except the constraints of form

$$\sum_{i \in L_1} x_i + \sum_{i \in L_2} x'_i + \sum_{i \in L_{12}} z_i > m \qquad \text{for every critical tuple } L_1, L_2, L_{12},$$

since there may be exponentially many critical tuples. Therefore the only challenge is whether any constraint of this form is violated. That is for given strategies $\mathbf{x}$ and $\mathbf{x}'$ and with $z_i = \max\{x_i, x'_i\}$, we need to design an algorithm that finds a critical tuple $\langle L_1, L_2, L_{12} \rangle$ (if any) for which we have

$$\sum_{i \in L_1} x_i + \sum_{i \in L_2} x'_i + \sum_{i \in L_{12}} \max\{x_i, x'_i\} \leq m. \tag{5.6}$$

For this, note that as implied by Observation 70, it suffices to be able to find a pure

strategy $\mathbf{y}$ of player 2 such that

$$\mathsf{u}_1(\mathbf{x}, \mathbf{y}) < u, \qquad \text{and,} \qquad \mathsf{u}_1(\mathbf{x}', \mathbf{y}) < u. \tag{5.7}$$

This is in some sense, equivalent to player 2's best-response which we show can be solved in polynomial time via a dynamic program. Define $D(j, m', \upsilon, \upsilon')$ to be 1 if and only if player 2 can use up to m' troops in the first j battlefields in a way that prevents $\mathbf{x}$ (resp. $\mathbf{x}'$) from obtaining a payoff of at least υ (resp. υ') in those battlefields. More precisely, $D(j, m', \upsilon, \upsilon')$ is 1 if and only if there exists a strategy $\mathbf{y}$ for player 2 such that

$$\sum_{i=1}^{j} y_i \leq m', \qquad \sum_{i:i\in[j],x_i>y_i} w_i \leq \upsilon, \qquad \text{and,} \qquad \sum_{i:i\in[j],x'_i>y_i} w_i \leq \upsilon'.$$

Clearly, if we are able to solve $D(j, m', \upsilon, \upsilon')$ for all possible inputs, then it suffices to check the value of $D(k, m, u, u)$ to see whether we can find a strategy $\mathbf{y}$ satisfying (5.7). Indeed, we can update the dynamic program in the following way:

$$D(j, m', \upsilon, \upsilon') = \max_{y_j\in\{0,\ldots,m'\}} D\Big(j-1, m'-y_j, \upsilon - g(y_j, j), \upsilon' - g'(y_j, j)\Big),$$

where,

$$g(y_j, j) = \begin{cases} w_j, \text{ if } x_j > y_j, \\ 0 \quad \text{otherwise,} \end{cases} \qquad \text{and,} \qquad g'(y_j, j) = \begin{cases} w_j, \text{ if } x'_j > y_j, \\ 0 \quad \text{otherwise.} \end{cases}$$

As for the base case, we set $D(0,0,0,0) = 1$. The correctness of the dynamic program is easy to confirm, since we basically check all possibilities for the number of troops that the second player can put on the jth battlefield and update the requirements on the previous battlefields accordingly. A minor issue, here, is that this only confirms whether a strategy $\mathbf{y}$ exists that satisfies (5.7) or not and does not give the actual strategy. However, one can simply obtain the actual strategy by slightly modifying the dynamic program to also store the strategy itself.

To summarize, we gave a polynomial time separating oracle for LP 5.5 that gives a polynomial time algorithm to solve it which leads to a $(u, 1/2)$-maximin 2-strategy for player 1 in polynomial. □

5.1.1.2 General Weights

Theorem 71 shows that the problem of computing a $(u, 1/2)$-maximin 2-strategy is computationally tractable when the weights are uniform. In this section, we study the general (i.e., non-uniform) setting and show that it is possible to obtain an (almost) optimal solution for this problem in polynomial time.

Recall that we assume there exists a $(u, 1/2)$-maximin 2-strategy and the goal is to either compute or approximate such a strategy. Fix the pure strategies of the solution to be $\hat{\mathbf{x}}$ and $\hat{\mathbf{x}}'$. Similar to Section 5.1.1.1, if we knew for which i's $\hat{x}_i \geq \hat{x}'_i$ holds and for which i's it is the opposite, we could model the problem as a linear program and find a solution in polynomial time. Since w_i's are not necessarily the same, unlike Section 5.1.1.1, we need to try an exponential number of combinations

to make a correct decision. To alleviate this problem, we show a generalized variant of the above argument. Define the status of battlefield i to be compatible with either $\leq$ or $\geq$ (or both in case of equality) based on the comparison of $\hat{x}_i$ and $\hat{x}'_i$. Assume we make a guess for the status of battlefields, which is incorrect for a set S of battlefields but correct for the rest of them. This means that for every battlefield i in set S, if $\hat{x}_i > \hat{x}'_i$, we assume $x_i \leq x'_i$ and vice versa. We show that if the total weight of the battlefields in S is small, there exists an almost optimal solution for the problem whose status is compatible with our guess.

For simplicity, we represent a guess for the status of the battlefields with a vector $\mathbf{g} \in \{\leq, \geq\}^k$ of length k in which every entry is either '$\leq$' or '$\geq$'. A solution $(\mathbf{x}, \mathbf{x}')$ is compatible with this guess if g_i correctly compares x_i to x'_i.

Lemma 72. *Let $(\hat{\mathbf{x}}, \hat{\mathbf{x}}')$ be an optimal (u, p)-maximin solution of the problem and $\mathbf{g}$ be a guess for the status of the battlefields. Let S be the set of battlefields for which $\mathbf{g}$ makes an incorrect comparison between $\hat{\mathbf{x}}$ and $\hat{\mathbf{x}}'$ on these battlefields. If $\sum_{i \in S} w_i = \alpha$ then there exists a $(u - \alpha, p)$-maximin strategy that is compatible with $\mathbf{g}$.*

Proof. We construct a pair of strategies $(\mathbf{x}, \mathbf{x}')$ based on $\hat{\mathbf{x}}$ and $\hat{\mathbf{x}}'$. For every battlefield $i \in S$, we set $x_i = x'_i = 0$ and for every battlefield $i \notin S$ we set $x_i = \hat{x}_i$ and $x'_i = \hat{x}'_i$. If a strategy of player 2 prevents both $\mathbf{x}$ and $\mathbf{x}'$ from getting a utility of $u - \alpha$, then the same strategy prevents both $\hat{\mathbf{x}}$ and $\hat{\mathbf{x}}'$ from getting a payoff of u. Therefore, $(\mathbf{x}, \mathbf{x}')$ is $(u - \alpha, p)$-maximin. Since for every battlefield outside set S we have $x_i = x'_i = 0$, then both $\leq$ and $\geq$ correctly compare the corresponding values

for such battlefields. Therefore, $(\mathbf{x}, \mathbf{x}')$ is compatible with $\mathbf{g}$. □

A simple interpretation of Lemma 72 is that if we make a guess that differs from a correct guess in a subset of battlefields with a total weight of α, we can use this guess to find a solution with an additive error of at most α in the utility. Based on this idea, we present a polynomial time algorithm that for any arbitrarily small constant $\epsilon < 1$ computes a $((1-\epsilon)u, 1/2)$-maximin 2-strategy.

δ-Uniform weights. One of the crucial steps of our algorithms, is updating battlefield weights. This step, is indeed used in multiple other places of this thesis as well. For a parameter δ, we define a *δ-uniform* variant of the game to be an instance on which the weight of each of the battlefields is *rounded down* to be in set $\mathcal{W} = \{1, (1+\delta)^1, (1+\delta)^2, \ldots\}$. That is, for any $i \in [k]$, we set the updated weight of battlefield i to be $w'_i = \max\{w : w \in \mathcal{W}, w \leq w_i\}$. The following observation implies that we can safely assume the game is played on the updated weights without losing a considerable payoff.

Observation 73. *For any u', any (u', p)-maximin strategy of the game instance $\mathcal{B}(n, m, \mathbf{w'})$ with the updated weights is a $((1-\delta)u', p)$-maximin strategy of the original instance $\mathcal{B}(n, m, \mathbf{w})$. Similarly, any (u', p)-maximin strategy of the original instance $\mathcal{B}(n, m, \mathbf{w})$ is a $((1-\delta)u', p)$-maximin for instance $\mathcal{B}(n, m, \mathbf{w}')$.*

Proof. Let $\mathbf{x} = (x_1, \ldots, x_k)$ be any pure strategy in the support of the (u', p)-maximin strategy of $\mathcal{B}(n, m, \mathbf{w'})$. Consider any arbitrary strategy $\mathbf{y}$ of player 2, it suffices to show that $\mathbf{x}$ gets a payoff of at least $(1-\delta)u'$ against $\mathbf{y}$ in the original instance $\mathcal{B}(n, m, \mathbf{w})$. Note that for any $i \in [k]$ we have $w'_i \geq w_i/(1+\delta) \geq (1-\delta)w_i$ by the

way that we round down the weights; therefore, we have that

$$\sum_{i:x_i>y_i} w'_i \geq \sum_{i:x_i>y_i} (1-\delta)w_i \geq (1-\delta) \sum_{i:x_i>y_i} w_i \geq (1-\delta)u',$$

completing the proof for the first part.

Similarly, since the weight of each battlefield is multiplied by a factor of at most $1/(1+\delta)$, any (u', p)-maximin strategy for the original instance is a $(u'/(1+\delta), p)$-maximin or simply a $((1-\delta)u', p)$-maximin strategy for instance $\mathcal{B}(n, m, \mathbf{w}')$.

□

The algorithm in a nutshell. We first update the weight of every battlefield i to be $\min\{u, w_i\}$. This, in fact, does not change the game instance for player 1 since his only objective is to guarantee a payoff of at least u. Now, for $\delta = \epsilon^3/10$ which is a relatively smaller error threshold than ϵ, we consider the δ-uniform variant of the game. In the δ-uniform instance, since the weights change exponentially in $1+\delta$, we have at most $O(1/\delta \cdot \log u)$ distinct weights . We put the battlefields with the same weight into a bucket and denote the sizes of the buckets by $k_1, k_2, \ldots, k_b$ where b is the number of buckets. Recall from Section 5.1.1.1 that if a set of battlefields have the same weight, then we are indifferent between these battlefields and thus the only information relevant to these battlefields is on how many of them $\hat{\mathbf{x}}$ puts more troops than $\hat{\mathbf{x}}'$. Therefore one way to make a correct guess is to try all $\prod(k_i+1)$ possibilities for all of the buckets. Unfortunately, $\prod(k_i+1)$ is not polynomial since the number of buckets is not constant. In order to reduce the number of possibilities

to a polynomial, we make a number of observations.

First, since the weights decrease exponentially between the buckets, the number of distinct weights that are larger than $\delta u/k$ (and smaller than u as described above,) is at most $\log_{1+\delta} k/\delta = O_\delta(\log k)$. Observe that we can safely ignore (i.e., make a wrong guess for) all the buckets with weight less than $\delta u/k$ since sum of the weights of all battlefields in such buckets is at most δu and by Lemma 72 it causes us to lose an additive error of at most δu. Although this reduces the number of buckets down to $O_\delta(\log k)$, it is still more than we can afford to try all $\prod(k_i + 1)$ possibilities.

Second, instead of trying $k_j + 1$ possibilities for bucket j, we reduce it down to only $O(1/\delta)$ options. More precisely, let t_j be the number of battlefields i in bucket j such that $\hat{x}_i \geq \hat{x}'_i$. For any bucket j with $k_j > 1/\delta$, if we only consider t_j to be in set $\{0, \lfloor \delta k_j \rfloor, \lfloor 2\delta k_j \rfloor, \lfloor 3\delta k_j \rfloor, \ldots, k_j\}$ one of the realizations of t_j makes at most $\delta k_j + 1$ incorrect guesses for bucket j. We use this later to argue that we do not lose a significant payoff by considering only $O(1/\delta)$ possibilities per bucket. As a result, we reduce the size of the cartesian product of all possibilities over all the buckets down to a polynomial.

Third, we show that if $n \geq (1+\epsilon)m/2$, we can safely assume that losing the value of at most $\lfloor \delta k_j \rfloor$ battlefields of buckets with more than $1/\delta$ battlefields does not hurt the payoff significantly. In other words, when $n \geq (1+\epsilon)m/2$, the optimal utility u is much larger than the total sum of the payoff we lose by only trying $1/\delta$ possibilities for every bucket j (proven in Lemma 74).

Based on the above ideas, we outline our PTAS as follows: (i) We first set a

cap of u for the weight of the battlefields. (ii) Let $\delta = \epsilon^3/10$. Next, we round down the weight of the battlefields to be powers of $(1+\delta)$. (iii) We put the battlefields with the same weights in the same bucket and remove the buckets whose battlefield weights are smaller than $\delta u/k$. (iv) Finally, we try $O(1/\delta)$ possibilities for the status of the battlefields within each bucket and check its feasibility using LP 5.5.[2] We return the first feasible solution that we find. The formal algorithm is given as Algorithm 9.

Algorithm 9: Algorithm to find a (u,p)-maximin 2-strategy for general weights.

Input: A payoff u for which existence of a $(u, 1/2)$-maximin 2-strategy is guaranteed.

Output: Two pure strategies $\mathbf{x}$ and $\mathbf{x}'$ that form a $(u, 1/2)$-maximin when played with equal prob. 1/2.

1: For every battlefield i, update w_i to be $\min\{u, w_i\}$.
2: For $\delta = \epsilon^3/10$, we further update the battlefield weights and consider its δ-uniform variant.
3: Ignore every battlefield i with weight less than $\delta u/k$ (i.e., naively guess $x_i \geq x'_i$).
4: Put all battlefields of the same weight into the same bucket and denote by k_j the number of battlefields in bucket j.
5: For each bucket j with $k_j \leq 1/\delta$, let $G_j = \{\leq, \geq\}^{k_j}$ be the set of all possible guesses for it.
6: For each bucket j with $k_j > 1/\delta$, let G_j be the set of all guesses where for any $d \in \{0, \lfloor \delta k_j \rfloor, \lfloor 2\delta k_j \rfloor, \lfloor 3\delta k_j \rfloor, \ldots, k_j\}$ we have $x_i \geq x'_i$ for any $i \leq d$ and $x_i \leq x'_i$ for any $i > d$.
7: Let $G = G_1 \times \ldots \times G_b$ be the cartesian product of the partial guesses of the buckets.
8: For any guess $\mathbf{g} \in G$, construct an instance of LP 5.5 and return the first found feasible solution $(\mathbf{x}, \mathbf{x}')$.

Before we present a formal proof, we state an auxiliary lemma to show a lower bound on the value of u when $n \geq (1+\epsilon)m/2$.

[2] As a minor technical detail, since our goal is to guarantee a payoff of at least $(1-\epsilon)u$ instead of u, we need to update the definition of losing tuples accordingly for this case.

Lemma 74. *Let α be the total sum of the weights of the battlefields whose buckets have a size of at least $1/\delta$. If $n > (1+\epsilon)m/2$ then there exists a $(\epsilon\alpha/8, 1/2)$-maximin strategy for player 1 that randomizes over two pure strategies.*

Proof. Let $\mathcal{B} = \{B_1, \ldots, B_b\}$ be the set of all buckets with at least $1/\delta$ battlefields. We slightly abuse the notation and respectively denote by w_i and k_i the weight and the number of battlefields in B_i. This means we have $\sum_{i=1}^{b} k_i w_i = \alpha$. We construct a 2-strategy $(\mathbf{x}, \mathbf{x}')$ where both $\mathbf{x}$ and $\mathbf{x}'$ are played with equal probability $1/2$ and claim that it is $(\epsilon\alpha/8, 1/2)$-maximin. To that end, for any bucket $B_i \in \mathcal{B}$ with an odd number of battlefields, we ignore one battlefield (i.e., we put zero troops in it in both $\mathbf{x}$ and $\mathbf{x}'$) to consider only an even number of battlefields for each bucket. Denote by α' the total weight of the remaining battlefields, i.e., the battlefields in some $B_i \in \mathcal{B}$ that are not ignored. It is easy to confirm that

$$\alpha' \geq (1-\delta)\alpha \tag{5.8}$$

since from each of the buckets in $\mathcal{B}$, at most one battlefield is ignored, which is only a δ fraction of the battlefields in that bucket since all buckets in $\mathcal{B}$ are assumed to have at least $1/\delta$ battlefields. Now, since only an even number of battlefields remain in each bucket B_i, we can partition them into two disjoint subsets $B_i^{(1)}$ and $B_i^{(2)}$ of equal size. Strategies $\mathbf{x}$ and $\mathbf{x}'$ are then constructed as follows:

- In strategy $\mathbf{x}$, for any $i \in [b]$, we put exactly $2w_i n/\alpha'$ troops in each battlefield in $B_i^{(1)}$. We put zero troops in all other battlefields.

- In strategy $\mathbf{x}'$, for any $i \in [b]$, we put exactly $2w_i n/\alpha'$ troops in each battlefield in $B_i^{(2)}$. We put zero troops in all other battlefields.

Observe that the total number of troops that we use in each of the strategies $\mathbf{x}$ and $\mathbf{x}'$ is exactly n as required. To see this, in strategy $\mathbf{x}$ for instance, the number of troops that are used is

$$\sum_{i\in[b]} \frac{2w_i n}{\alpha'}|B_i^{(1)}| = \frac{2n\sum_{i\in[b]} w_i|B_i^{(1)}|}{\alpha'} = \frac{2n\sum_{i\in[b]} w_i|B_i^{(1)}|}{\sum_{i\in[b]} w_i \cdot 2|B_i^{(1)}|} = n.$$

It only remains to prove that this strategy is $(\epsilon\alpha/8, 1/2)$-maximin. Assume for the sake of contradiction that player 2 has a strategy $\mathbf{y}$ that prevents both $\mathbf{x}$ and $\mathbf{x}'$ from achieving a payoff of at least $\epsilon\alpha/8$. We can assume w.l.o.g., that for every battlefield i, we have $y_i \in \{0, x_i, x_i'\}$, thus, on all battlefields that are ignored (i.e., $x_i = x_i' = 0$), we have $y_i = 0$. Further, note that because of the special construction of strategies $\mathbf{x}$ and $\mathbf{x}'$, in each battlefield, at most one of $\mathbf{x}$ or $\mathbf{x}'$ put non-zero troops; therefore, on every battlefield i where $y_i > 0$, either we have $x_i > 0$ or $x_i' > 0$. Define

$$m_1 = \sum_{i\in[k]:x_i>0} y_i, \qquad \text{and,} \qquad m_2 = \sum_{i\in[k]:x_i'>0} y_i.$$

Note that $m_1 + m_2 \leq m$ since it cannot be the case that both x_i and x_i' are non-zero at the same time as described above. Therefore at least one of m_1 or m_2 is not more than $m/2$. Assume w.l.o.g., that $m_1 \leq m/2$. One can think of m_1 as the number of troops that are spent by player 2 to prevent strategy $\mathbf{x}$ from getting a payoff of at least $\epsilon\alpha/8$. To obtain the contradiction, we prove that player 2 cannot use only

$m/2$ troops to prevent $\mathbf{x}$ from obtaining a payoff of $\epsilon\alpha/8$. Recall that we denote the total sum of battlefields on which we put a non-zero number of troops either in $\mathbf{x}$ or $\mathbf{x}'$ by α'. Strategy $\mathbf{x}$ puts non-zero troops in half of these battlefields, and therefore sum of their weights is at least $\alpha'/2$. To prevent $\mathbf{x}$ from getting a payoff of at least $\epsilon\alpha/8$, player 2 can lose a weight of less than $\epsilon\alpha/8$ on these battlefields. Let $\mathcal{Y}$ be the subset of battlefields on which $\mathbf{y}$ wins $\mathbf{x}$ and let $w(\mathcal{Y})$ be the total weight of all these battlefields. We need to have

$$w(\mathcal{Y}) > \alpha'/2 - \epsilon\alpha/8. \tag{5.9}$$

Since the number of troops that is put on the battlefields in $\mathbf{x}$ is proportional to the battlefield weights on which $\mathbf{x}$ puts non-zero troops, we have

$$\sum_{i\in\mathcal{Y}} x_i \geq \Big(\frac{w(\mathcal{Y})}{\alpha'/2}\Big)n \overset{\text{By (5.9)}}{\geq} \Big(\frac{\alpha'/2-\epsilon\alpha/8}{\alpha'/2}\Big)n \geq \Big(1-\frac{\epsilon\alpha}{4\alpha'}\Big)n \overset{\text{By (5.8)}}{\geq} \Big(1-\frac{\epsilon}{4(1-\delta)}\Big)n \overset{\text{Since } \delta=\epsilon^3/10}{\geq} \frac{n}{1+\epsilon}.$$

Therefore, to be able to match the troops of $\mathbf{x}$ in every battlefield in $\mathcal{Y}$, using only $m/2$ troops, we have

$$m/2 \geq \sum_{i\in\mathcal{Y}} x_i \geq \frac{n}{1+\epsilon}. \tag{5.10}$$

The last inequality contradicts the assumption of the lemma that $n > (1+\epsilon)m/2$. Therefore, there exists no such strategy $\mathbf{y}$ for player 2. That means, the constructed strategy is indeed a $(\epsilon\alpha/8, 1/2)$-maximin 2-strategy. □

Theorem 75. *Let $\epsilon > 0$ be an arbitrarily small constant and suppose that we have an instance of continuous Colonel Blotto in which $n \geq (1+\epsilon)m/2$. Given that*

player 1 has a $(u, 1/2)$-*maximin* 2-*strategy, there exists an algorithm that finds a* $((1-\epsilon)u, 1/2)$-*maximin* 2-*strategy for him in polynomial time.*

Proof. The algorithm to achieve this strategy is given as Algorithm 9. We first analyze the approximation factor of Algorithm 9 and then prove that it runs in polynomial time.

Approximation factor. Let $\mathbf{w}'$ denote the updated battlefield weights by the end of Line 2 of Algorithm 9. First note that setting a cap of u for the battlefield weights in the first line of algorithm does not change the game instance at all since the only goal of player 1 is to guarantee a payoff of at least u. Second, we know by Observation 73 that for any u', any (u', p)-maximin strategy for the δ-uniform variant is a $((1-\delta)u', p)$-maximin strategy for the original (i.e., not δ-uniform) instance. Roughly speaking, since δ is relatively smaller than ϵ, we still get a $((1-\epsilon)u, 1/2)$-maximin strategy for the original instance if we achieve a good approximation on the δ-uniform variant.

Consider an optimal $(u, 1/2)$-maximin 2-strategy for player 1 on the original instance (which recall is guaranteed to exist) and assume that it randomizes over two pure strategies $\hat{\mathbf{x}}$ and $\hat{\mathbf{x}}'$. By Observation 73, this is a $((1-\delta)u, 1/2)$-maximin strategy for the δ-uniform variant. Let us denote by vector $\mathbf{c} \in \{\leq, \geq\}^k$ the comparison between the entries of $\hat{\mathbf{x}}$ and $\hat{\mathbf{x}}'$. That is, c_i is '$\geq$' if and only if $\hat{x}_i \geq \hat{x}'_i$ and it is '$\leq$' otherwise. Our goal is to argue that there exists a guess $\mathbf{g} \in G$ that is *sufficiently close* to $\mathbf{c}$ — where by sufficiently close we mean sum of weights of all battlefields with $g_i \neq c_i$ is very small. We will later combine this with Lemma 72 to obtain the

desired guarantee.

We first assume w.l.o.g. that for any two battlefields i and j with $i \leq j$ that have the same weight, if $c_i \neq c_j$, then $c_i =$ '$\leq$' and $c_j =$ '$\geq$' (otherwise we swap these two battlefields without changing the payoff guaranteed by the strategy). We have two relaxations over the guesses in Lines 3 and 6 of Algorithm 9.

First, in Line 3 of Algorithm 9, sum of weights of all light battlefields with weight at most $\delta u/k$ is not more than δu since k is the total number of battlefields. Thus, even if $g_i \neq c_i$ on these battlefields, their total sum is less than δu.

Second, in Line 6 of Algorithm 9, let us denote by α the sum of weights of all battlefields whose bucket contains more than $1/\delta$ battlefields. Observe that we check almost all possibilities of guesses on these buckets, except on δ fraction of their battlefields. More precisely, the total sum of weights of such battlefields on which our guess is wrong is at most $\delta\alpha$. It only remains to argue that $\delta\alpha$ cannot be very large. Note that by Lemma 74, we can obtain a simple $(\epsilon\alpha/8, 1/2)$-maximin strategy since the condition of $n \geq (1+\epsilon)m/2$ is also satisfied here. Thus, we can assume $u \geq \epsilon\alpha/8$ or simply $\alpha \leq 8u/\epsilon$ (otherwise instead of Algorithm 9 we return the strategy of Lemma 74). Therefore the total weight of battlefields of this type, on which we guess wrong is no more than

$$\delta\alpha \leq \frac{\delta 8u}{\epsilon} \overset{\text{Since } \delta = \epsilon^3/10}{\leq} \frac{4}{5}\epsilon^2 u.$$

Combining these two, we show that there exists a guess $\mathbf{g} \in G$ for which sum

of battlefields with $g_i \neq c_i$ is at most

$$\delta u + \frac{4}{5}\epsilon^2 u = \frac{1}{10}\epsilon^3 u + \frac{4}{5}\epsilon^2 u < \epsilon^2 u.$$

By Lemma 72, this implies a $((1-\delta)u - \epsilon^2 u, 1/2)$-maximin 2-strategy for the δ-uniform variant; and by Observation 73, guarantees a utility of at least

$$(1-\delta-\epsilon^2)(1-\delta)u = (1-\frac{\epsilon^3}{10}-\epsilon^2)(1-\frac{\epsilon^3}{10})u \geq (1-\epsilon)u$$

for the original instance with probability at least $1/2$, which in other words, gives a $((1-\epsilon)u, 1/2)$-maximin 2-strategy as desired.

Running time. It is easy to confirm that the running time of Algorithm 9 is poly($|G|$). Thus, it suffices to show that the total number of guesses in G is polynomial. Observe that for any $i \in [b]$, the total number of partial guesses for bucket i is $O(1)$ (though dependent on δ). On the other hand, the total number of buckets as mentioned before is at most $O(\log k)$ (we hide the dependence on δ) therefore $|G| \leq O(1)^{O(\log k)}$ which is polynomial. □

Notice that when $n \leq m/2$, there is no chance for player 1 to get a nonzero utility by randomizing over two pure strategies since player 2 can always play $y_i = \max\{x_i, x_i'\}$ troops on every every battlefield and win all of them.

By generalizing the ideas mentioned above, we can extend our approach to the case when $c > 2$. However, the core ideas behind the generalization are mentioned above.

5.2 Probability Distribution Over the Support

In this section we discuss the structure of probabilities that are assigned to the pure strategies in the support of an optimal (u, p)-maximin strategy. It is worth mentioning that our results in this section apply to any two player game so long as the goal is to compute a (u, p)-maximin strategy.

Recall that given a minimum utility u, and a given upper bound c on the support size, our goal is to compute a (u, p)-maximin c-strategy for maximum possible p; we call this an optimal (u, p)-maximin c-strategy or simply an optimal strategy when it is clear from context. Naively, there are uncountably many ways to assign probabilities to the c pure strategies in the support. However, in this section we show that there are a small number of (efficiently constructible) possibilities for the probabilities among which an optimal solution is guaranteed to exist.

Definition 76. *For a mixed strategy* $\mathbf{x}$*, define the profile* $\rho(\mathbf{x})$ *of* $\mathbf{x}$ *to be the multiset of probabilities associated to the pure strategies in the support of* $\mathbf{x}$*.*

The main theorem of this section is the following.

Theorem 77. *For every constant* $c > 0$*, there exists an algorithm to construct a set* P_c *of* $O(1)$ *profiles in time* $O(1)$*, such that existence of an optimal* (u, p)*-maximin* c*-strategy* $\mathbf{x}$ *with* $\rho(\mathbf{x}) \in P_c$ *is guaranteed.*

As a corollary of Theorem 77, in order to find an optimal (u, p)-maximin c-strategy, one can iterate over the profiles in P_c, optimize the strategy based on each profile, and report the best solution.

Let us start with the case when $c = 1$. Clearly, in this case, we have no choice but to play our pure strategy with probability 1, therefore it suffices to have profile $\{1\}$ in set P_1, i.e., $P_1 = \{\{1\}\}$. For the case of $c = 2$, we play two pure strategies with probabilities q and $q' = 1 - q$. Our claim for this case, is that it suffices to have $P_2 = \{\{1\}, \{1/2, 1/2\}\}$.[3] Note that if the profile of a 2-strategy $\mathbf{x}$ is $\{1\}$ it is also a 1-strategy since it plays only 1 pure strategy with non-zero probability.

To prove that $P_2 = \{\{1\}, \{1/2, 1/2\}\}$ is sufficient, we first show that if a 2-strategy is (u, p)-maximin for some $p > 1/2$, then there also exists a $(u, 1)$-maximin 1-strategy, thus, $\{1\} \in P_2$ suffices.

Observation 78. *If player 1 has a (u, p)-maximin 2-strategy with $p > 1/2$, then he also has a $(u, 1)$-maximin 1-strategy.*

Proof. Let $\mathbf{x}$ and $\mathbf{x}'$ be the two strategies in the support of a given (u, p)-maximin 2-strategy for some $p > 1/2$ and assume w.l.o.g., that strategy $\mathbf{x}$ is played with probability at least $1/2$ (this should be true for at least one of the strategies in the support of any 2-strategy). Since $p > 1/2$, by definition of a (u, p)-maximin strategy, $\mathbf{x}$ should obtain a payoff of at least u against any strategy of player 2. This means that pure strategy $\mathbf{x}$ itself, is a $(u, 1)$-maximin 1-strategy. □

On the other hand, the following observation shows that for an optimal (u, p)-maximin 2-strategy, we have $p \geq 1/2$.

Observation 79. *If player 1 has a (u, p)-maximin 2-strategy $\mathbf{x}$ for some $0 < p < 1/2$,*

[3]Note that the elements in P_c are multisets, hence multiple occurrences of the same element is allowed in them.

then by playing each of the two pure strategies of $\mathbf{x}$ *with probability* $1/2$*, he gets a better* $(u, 1/2)$-*maximin* 2-*strategy.*

Proof. Let $\mathbf{x}^1$ and $\mathbf{x}^2$ be the two pure strategies in the support of $\mathbf{x}$. Since $p > 0$, against any pure strategy $\mathbf{y}$ of the opponent, at least one of $\mathbf{x}^1$ or $\mathbf{x}^2$ obtain a utility of at least u. Thus, if they are both played with probability $1/2$, the resulting strategy is $(u, 1/2)$-maximin. □

Combining the two observations above, we can conclude that for an optimal (u, p)-maximin 2-strategy, either $p = 1$ or $p = 1/2$. For the former case Observation 78 implies that $\{1\} \in P_2$ is sufficient and for the latter $\{1/2, 1/2\} \in P_2$ is sufficient by Observation 79. We remark that it is necessary for any choice of P_2 to have both $\{1\}$ and $\{1/2, 1/2\}$. Therefore our choice of $P_2 = \{\{1\}, \{1/2, 1/2\}\}$ is both sufficient and necessary.

We do not attempt to prove it here, as it is not required for the proof of our main theorem in this section, however, it can also be shown that for the case of 3-strategies, it suffices to have

$$P_3 = \{\{1\}, \{1/2, 1/2\}, \{1/3, 1/3, 1/3\}\}.$$

Unfortunately, the case of $c \geq 4$ does not follow the same pattern. That is, e.g. when $c = 4$, it is not sufficient to have $P_4 = \big\{\{1\}, \{1/2, 1/2\}, \{1/3, 1/3, 1/3\}, \{1/4, 1/4, 1/4, 1/4\}\big\}$. An example was given for this in Table 5.1 in the context of the Colonel Blotto game where the optimal 4-strategy has profile $\{2/5, 1/5, 1/5, 1/5\}$. Nevertheless, we show that

size of P_c, for any constant c, can be bounded by a constant.

Consider a (u,p)-maximin c-strategy $\mathbf{x}$ for player 1 in a two player game $\mathcal{G}$ and assume w.l.o.g. that the strategies in its support are numbered from 1 to c. A subset $W \subseteq [c]$ is a *u-winning-subset* of $\mathbf{x}$, if player 2 has a strategy $\mathbf{y}$ such that the ith strategy in the support of $\mathbf{x}$ gets a payoff of at least u against $\mathbf{y}$ if and only if $i \in W$. We denote by $\mathsf{W}_u(\mathbf{x})$ the set of all u-winning-subsets of $\mathbf{x}$. The following claim implies that it suffices to have the set $\mathsf{W}_u(\mathbf{x})$ of any strategy $\mathbf{x}$ to decide how to assign probabilities to the strategies in its support.

Lemma 80. *Given the set $\mathsf{W}_u(\mathbf{x})$ of a (u,p)-maximin c-strategy $\mathbf{x}$, one can, in time $O(1)$, assign probabilities $\rho_1, \ldots, \rho_c$ to the pure strategies in the support of $\mathbf{x}$ in a way that the modified strategy $\mathbf{x}'$ is (u,p')-maximin for some $p' \geq p$.*

Proof. Let $\mathcal{W}$ be the given set. Clearly we have $|\mathcal{W}| \leq O(1)$ since it is a subset of the power set of $[c]$ and c is a constant. The following linear program finds $\rho_1, \ldots, \rho_c$ in time $O(1)$ since it has $O(1)$ variables and $O(1)$ constraints. We show that by playing the ith pure strategy in the support of $\mathbf{x}$ with probability ρ_i, we obtain a strategy $\mathbf{x}'$ that is as good as $\mathbf{x}$.

$$\begin{array}{lll} \text{maximize} & p' & \\ \text{subject to} & \rho_i \geq 0 & \forall i \in [c] \\ & \sum \rho_i = 1 & \\ & p' \leq \sum_{i \in W} \rho_i & \forall W \in \mathcal{W} \end{array} \tag{5.11}$$

Let us denote by q_i the actual probability with which the ith strategy in the support

of $\mathbf{x}$ is played. Observe that the probability p for which strategy $\mathbf{x}$ guarantees receiving a payoff of at least u is exactly $\min_{W \in \mathsf{W}_u(\mathbf{x})} \sum_{i \in W} q_i$. This comes from the fact that by definition if W is a u-winning-subset of $\mathbf{x}$, then player 2 has a strategy $\mathbf{y}$ against which player 1 receives a payoff of u by its ith pure strategy iff $i \in W$. This means the probability of guaranteeing a utility of u against $\mathbf{y}$ is equal to $\sum_{i \in W} q_i$. On the other hand, by definition of (u,p)-maximin, we need to guarantee a payoff of at least u with probability at least p against *any* strategy of the opponent. Therefore we have $p = \min_{W \in \mathsf{W}_u(\mathbf{x})} \sum_{i \in W} q_i$. Now, observe that LP 5.11 finds probabilities ρ_i in a way that precisely maximizes $\min_{W \in \mathsf{W}_u(\mathbf{x})} \sum_{i \in W} \rho_i$. Therefore, if we play the ith pure strategy in the support of $\mathbf{x}$ with probability ρ_i, we obtain a strategy that is as good as $\mathbf{x}$. □

Claim 81. *One can obtain a set S of size $O(1)$ in time $O(1)$ such that for some optimal (u,p)-maximin c-strategy $\mathbf{x}$, we have $\mathsf{W}_u(\mathbf{x}) \in S$.*

Proof. Let S be the power set of the power set of $C = \{1, \ldots, c\}$. Note that $|S| = 2^{2^c} = O(1)$; thus, we can simply construct S in time $O(1)$. On the other hand, take an optimal (u,p)-maximin c-strategy $\mathbf{x}$. By definition each u-winning-subset of $\mathbf{x}$ is a subset of C and, thus, $\mathsf{W}_u(\mathbf{x})$ is a set of some subsets of C, thus, $\mathsf{W}_u(\mathbf{x}) \in S$ as desired. □

Indeed the two claims above are sufficient to prove Theorem 77.

Theorem 77 (restated). *For every constant $c > 0$, there exists an algorithm to construct a set P_c of $O(1)$ profiles in time $O(1)$, such that existence of an optimal (u,p)-maximin c-strategy $\mathbf{x}$ with $\rho(\mathbf{x}) \in P_c$ is guaranteed.*

Proof. Consider the set S provided by Claim 81. For every set $\mathcal{W} \in S$, construct a profile ρ using the algorithm of Lemma 80 and put it in a set P. Observe that $|P| = |S| = O(1)$. Therefore it only remains to prove that the profile of at least one optimal strategy is in P.

Note that by Claim 81, at least one set $\mathcal{W} \in S$ is the set of all u-winning-subsets of an optimal strategy. By Lemma 80, the constructed profile for $\mathcal{W}$ is the profile of a strategy that is also optimal. Therefore among the profiles in P, we have the profile of at least one optimal strategy which concludes the proof. □

5.3 Continuous Colonel Blotto

5.3.1 Generalization to the Case of c-Strategies for $c > 2$

In this section we generalize our results for the continuous variant to the case of multiple (i.e., more than 2) strategies in the support. That is, for a given u, we seek to find a (u, p)-maximin c-strategy $\mathbf{x}$ for maximum possible p. Throughout the section, we denote the support of $\mathbf{x}$ by $\mathbf{x}^1, \ldots, \mathbf{x}^c$.

We showed that the only computationally challenging problem for the case of 2-strategies is when our goal is to find $(u, 1/2)$-maximin strategies. For that we only needed to give two pure strategies $\mathbf{x}^1$ and $\mathbf{x}^2$ and make sure that against every pure strategy of the opponent, at least one of these strategies obtains a utility of at least u. This structure becomes more complicated when we allow more than 2 strategies in the support. Consider, for instance, the case of 3-strategies and suppose for simplicity that we are promised that the three pure strategies in the support of

an optimal strategy are each played with probability ⅓. For this example, the computationally challenging cases are obtaining either a $(u, 2/3)$-maximin strategy or a $(u, 1/3)$-maximin strategy. For the former case, we need to make sure that against every strategy of the opponent at least 2 of the strategies in the support obtain a utility of at least u. For the latter it suffices for 1 of the strategies to obtain a utility of at least u. The idea is to first attempt to find a $(u, 2/3)$-maximin. It could be the case that no such strategy exists; if so, we then attempt to find a $(u, 1/3)$-maximin strategy.

To generalize this to more than 3 strategies, we use the notion of *non-losing sets*.

Definition 82 (Non-losing sets)**.** *Consider a (u, p)-maximin c-strategy $\mathbf{x}$ with support $\mathbf{x}^1, \ldots, \mathbf{x}^c$. We define a set $N \in [c]$ to be a* non-losing set *of $\mathbf{x}$ if for every strategy $\mathbf{y}$ of player 2 there exists some $i \in N$ for which $\mathsf{u}_1(\mathbf{x}^i, \mathbf{y}) \geq u$. A set $N \in [c]$ is a* minimal non-losing *set of $\mathbf{x}$ if it is a non-losing set of $\mathbf{x}$ and there is no strict subset $N' \subsetneq N$ of N that is a non-losing set of $\mathbf{x}$. We denote the set of all minimal non-losing sets of a strategy $\mathbf{x}$ by $\mathcal{N}(\mathbf{x})$.*

Observe that for a $(u, 1/2)$-maximin 2-strategy, its only minimal non-losing set is $\{1, 2\}$. Moreover, for our example of a (u, p)-maximin 3-strategy, if $p = 2/3$, the minimal non-losing sets are $\{1, 2\}, \{1, 3\}, \{2, 3\}$. In the same example, if $p = 1/3$, the only minimal non-losing set is $\{1, 2, 3\}$.

The general structure of our algorithm is to first guess the set of all minimal non-losing sets $\mathcal{N}$ and see whether it is possible to *satisfy* it by constructing the

pure strategies $\mathbf{x}^1, \ldots, \mathbf{x}^c$ in such a way that we get $\mathcal{N}(\mathbf{x}) = \mathcal{N}$. Recall that there are only a constant number of possibilities for the choice of $\mathcal{N}$ since it is a subset of the power set of $[c]$ and c is constant. We also know by Lemma 80 that having $\mathcal{N}(\mathbf{x})$ is sufficient to decide what is the best way to assign probabilities to strategies $\mathbf{x}^1, \ldots, \mathbf{x}^c$ as it uniquely determines the winning subsets. Therefore we can find the optimal c-strategy if for a given choice of $\mathcal{N}$ we can decide in polynomial time whether it is satisfiable or not.

Now, given that our guess $\mathcal{N}$ is fixed, we need to find strategies $\mathbf{x}^1, \ldots, \mathbf{x}^c$ in such a way that every $N \in \mathcal{N}$ is indeed a nonlosing set. This gives the following formulation of the problem.

$$
\begin{array}{lll}
\text{find} & \mathbf{x}^1, \ldots, \mathbf{x}^c & \\
\text{subject to} & x_i^j \geq 0 & \forall i, j : i \in [k], j \in [c] \\
& \sum_{i \in [k]} x_i^j \leq n & \forall j \in [c] \\
& \text{for some } i \in N \text{ we have } \mathsf{u}_1(\mathbf{x}^i, \mathbf{y}) \geq u & \forall \mathbf{y} \in \mathcal{S}_2, \forall N \in \mathcal{N}
\end{array}
\tag{5.12}
$$

Program 5.12 is in fact the generalization of Program 5.3 to the case of c-strategies. Clearly, the last constraint in its current form is not linear. We showed how Program 5.3 can be decomposed into a set of convex polytopes and how each of them can be solved in polynomial time via the definition of critical tuples. We follow a similar approach and give a generalized definition of critical tuples.

Definition 83 (Critical tuples)**.** *Consider a tuple* $\mathbf{W} = (W_1, \ldots, W_k)$ *where each* W_i *is a subset of* $[c]$*. We call* $\mathbf{W}$ *a* critical tuple *if and only if for some* $N \in \mathcal{N}$ *we*

have

$$\sum_{i:j\in W_i} w_i < u \qquad\qquad \forall j \in N.$$

Consider a tuple $\mathbf{W} = (W_1, \ldots, W_k)$. Fix a strategy $\mathbf{y}$ of player 2, and assume $\mathbf{x}^1, \ldots, \mathbf{x}^c$ are constructed in such a way that $x_i^j > y_i$ iff $j \in W_i$. In other words, W_i is the set of indices of the strategies that win the ith battlefield against $\mathbf{y}$. Now $\mathbf{W}$ is a critical tuple iff $\mathbf{x}^1, \ldots, \mathbf{x}^c$ do not satisfy $\mathcal{N}$. More precisely, $\mathbf{W}$ is a critical tuple iff all of the strategies in one of the minimal non-losing sets of $\mathcal{N}$ lose (i.e., get a payoff of less than u) against $\mathbf{y}$.

Before describing how we can use critical tuples in rewriting Program 5.12, we need to know precisely how for each $i \in [k]$, the values of $x_i^1, x_i^2, \ldots, x_i^c$ compare to each other. To that end, we define a *configuration* $\mathbf{G}$ to be a vector of k matrices $G_1, \ldots, G_k$ which we call *partial configurations*, where for any $i \in [k]$, and for any $j_1, j_2 \in [c]$, the value of $G_i(j_1, j_2)$ is '$\leq$' if $x_i^{j_1} \leq x_i^{j_2}$ and it is '$\geq$' otherwise.

Clearly if we fix the configuration $\mathbf{G}$ of strategies a priori, it is possible to ensure the found strategies comply with it via $O(kc^2)$ linear constraints. It suffices to have one constraint for every $G_i(j_1, j_2)$. The linear program below shows that if the configuration is fixed, we can even rewrite Program 5.12 as a linear program. For configuration $\mathbf{G}$ and critical tuple $\mathbf{W}$, define $z_i(\mathbf{G}, \mathbf{W}) := \arg\max_{j:j\notin W_i} x_i^j$. Note that it is crucial that $z_i(\mathbf{G}, \mathbf{W})$ is solely a function of $\mathbf{G}$ and $\mathbf{W}$ (and not the actual

strategies $\mathbf{x}^1, \ldots, \mathbf{x}^c$) so long as strategies $\mathbf{x}^1, \ldots, \mathbf{x}^c$ comply with $\mathbf{G}$.

$$
\begin{array}{lll}
\text{find} & \mathbf{x}^1, \ldots, \mathbf{x}^c & \\
\text{subject to} & x_i^j \geq 0 & \forall i, j : i \in [k], j \in [c] \\
& \sum_{i \in [k]} x_i^j \leq n & \forall j \in [c] \\
& \text{ensure that } \mathbf{x}^1, \ldots, \mathbf{x}^c \text{ comply with } \mathbf{G} & \\
& \sum_{i \in [k]} x_i^{z_i(\mathbf{G}, \mathbf{W})} > m & \text{for every critical tuple } \mathbf{W} = (W_1, \ldots, W_k)
\end{array}
\tag{5.13}
$$

The intuition behind the last constraint is that for player 2 to be able to enforce any critical tuple to happen, he needs to have more than m troops. This is a sufficient and necessary condition to ensure that all of the non-losing sets in $\mathcal{N}$ are satisfied. Observe that if our guess for $\mathcal{N}$ is wrong, for every configuration $\mathbf{G}$, LP 5.13 will be infeasible.

The takeaway from LP 5.13, is that if we fix the configuration $\mathbf{G}$, the solution space becomes convex and can be described via linear constraints. Although there may be exponentially many critical tuples $\mathbf{W}$ and, thus, exponentially many constraints in LP 5.13, one can design an appropriate separating oracle and use ellipsoid method to solve it in polynomial time using a dynamic programming approach similar to the one used for solving LP 5.5. Therefore one algorithm to solve the problem is to iterate over all possible configurations, solve LP 5.13 for each and report the best solution. This gives us an exponential time algorithm with running time $(c!)^k$. Recall that we followed a rather similar approach for the case of 2-strategies and showed that the solution space can be decomposed to 2^k convex polytopes. To

overcome this when $c = 2$, we showed how it is possible to only consider polynomially many such polytopes as far as the problem is concerned. We follow the same approach here.

Let us start with uniform setting where all the battlefield weights are the same. Since the players are both indifferent to the battlefields in the uniform setting, instead of individually fixing the partial configuration of each battlefield, it only suffices to know the number of battlefields having a particular partial configuration. Note that each partial configuration, e.g., G_1, is determined uniquely if we are given the sorted order of $x_1^1, x_1^2, \ldots, x_1^c$. Therefore, there are a total number of $c! = O(1)$ possibilities for each partial configuration. This means, if we only count the number of battlefields having each partial configuration, we reduce the total number of considered configurations down to a polynomial ($O(k^{c!})$ to be more precise). This gives a polynomial time algorithm for the uniform case.

Generalization to the case where the weights are not are equal follows from similar ideas described in Section 5.1.1.2. That is, we can consider the δ-uniform variant of the game for a relatively smaller error threshold than ϵ and group battlefields into buckets with each bucket containing the battlefields of the same updated weight. Then in each bucket, similar to the uniform case, it suffices to only count the number of battlefields of each configuration. Recall, however, that the crucial property for this idea to work was to show that for each bucket, it suffices to check only a constant number of different possibilities. Naively, the number of possibilities for each bucket is $k_i^{O(1)}$. Therefore, if $k_i \leq 1/\delta$ it is bounded by $O(1)$ as desired. However, if $k_i > 1/\delta$, the idea, similar to Section 5.1.1.2, is to discretize

the number of battlefields having one particular partial configuration of G_j to be in set $\{0, \delta k_i, 2\delta k_i, \ldots, k_i\}$. This way, a similar argument as in Lemma 74 shows that making a mistake on only a δ fraction of buckets with at least $1/\delta$ battlefields is negligible so long as our goal is to guarantee a utility of $(1-\epsilon)u$, resulting in the following theorem.

Theorem 84. *Let $\epsilon > 0$ be an arbitrarily small constant. Given that player 1 in an instance of continuous Colonel Blotto has a (u,p)-maximin c-strategy for a constant c, and given that $n \geq (1+\epsilon)m/\lfloor(1-p)c+1\rfloor$, there exists an algorithm that finds a $((1-\epsilon)u,p)$-maximin c-strategy for him in polynomial time.*

We remark that if $n \leq m/\lfloor(1-p)c+1\rfloor$, then player 1 has no (u,p)-maximin strategy.

5.4 Discrete Colonel Blotto

In this section we focus on the discrete variant of Colonel Blotto. Though similar in spirit to the continuous variant, discrete Colonel Blotto is an inherently different game (especially from a computational perspective) and requires different techniques.

5.4.1 The Case of One Strategy

Similar to the continuous variant, we start with the case where the support size is bounded by 1. We showed how for the continuous case, it is possible to obtain an optimal $(u,1)$-maximin strategy by solving LP 5.2. Observe, however, that variable

x_i in LP 5.2 denotes the number of troops in battlefield i; this implies that LP 5.2 relies crucially on the fact that a fractional number of troops in a battlefield is allowed. As such, the same idea cannot be applied to the discrete case since the integer variant of LP 5.2 is not necessarily solvable in polynomial time.

Prior algorithm of [3] gives a 2-approximation for this problem, i.e., given that there exists a $(u, 1)$-maximin strategy for player 1, they give a $(u/2, 1)$-maximin strategy in polynomial time. We improve this result by obtaining an almost optimal solution in polynomial time.

Theorem 85. *Given that player 1 has a $(u, 1)$-maximin strategy, there exists a polynomial time algorithm that obtains a $((1 - \epsilon)u, 1)$-maximin strategy of player 1 for any arbitrarily small constant $\epsilon > 0$.*

The algorithm in a nutshell. The algorithm that we use to prove Theorem 85 is composed of three main steps. In step 1, we round down the battlefield weights to be powers of $(1 + \epsilon/2)$, i.e., we consider the $(\epsilon/2)$-uniform variant of the game. The goal here is to reduce the number of distinct battlefield weights. As it was previously shown, by optimizing over this updated instance of the game, player 1 does not lose a considerable payoff on the original instance.

In step 2, we partition the battlefields into two subsets: *light* battlefields that have a weight of at most $(\epsilon/2)u$ and *heavy* battlefields that each has a weight of more than $(\epsilon/2)u$. Roughly speaking, the goal is to separate battlefields that have a high impact on the outcome of the game from the lower weight ones. We show that as a result of step 1, we can give a new representation for strategies of player 1

that reduces the total number of *partial strategies* of player 1 on heavy battlefields down to a polynomial. We further show that for any strategy of player 1 on heavy battlefields, player 2 has only a constant number of *valid* responses as far as the optimal solution is concerned. Importantly, bounding the number of responses of player 2 by a constant has a crucial role in solving the problem in polynomial time — we will elaborate more on this in the next paragraph.

In step 3, we propose a *weaker adversary* than player 2. Roughly speaking, we assume that for any given strategy of player 1, the weaker adversary responds greedily on light battlefields (though we do not limit him on heavy battlefields). We show that optimizing player 1's strategy against this weaker adversary guarantees an acceptable payoff against the actual adversary (i.e., player 2). In brief, the main advantage of optimizing player 1's strategy against the weaker adversary is in that it allows us to exploit the more predictable greedy response of the opponent. Recall, however, that we do not limit the weaker adversary on heavy battlefields and, therefore, his response to a strategy that we give for player 1 may still come from a somewhat unpredictable function. However, step 2 guarantees that for every strategy of player 1, it suffices to consider only a constant number of responses of the weaker adversary. This allows us to use a dynamic program that has, roughly, the same number of dimensions as the number of strategies of the weaker adversary and solve the problem in polynomial time.

Basic structural properties. Recall that a $(u, 1)$-maximin strategy guarantees a payoff of u against *any* strategy of player 2. Therefore, if $u > 0$, we need to have

$n > m$; or otherwise, no matter what pure strategy the first player chooses, the second player can match the number of troops of the first player in all battlefields and win them all. Now, assuming that $n > m$, if there exists one battlefield with weight at least u, the first player can simply put all his troops in that battlefield and guarantee a payoff of at least u. Therefore we assume throughout the rest of the section that $n > m$ and that the maximum battlefield weight is less than u.

Step 1: Updating the battlefield weights. We first *round down* the weight of each of the battlefields to be in set $\mathcal{W} = \{1, (1+\epsilon/2)^1, (1+\epsilon/2)^2, \ldots\}$ and denote the modified weights vector by $\mathbf{w}'$. That is, we consider the $(\epsilon/2)$-uniform variant of the game (see Section 5.1 and Observation 73 for the formal definition).

Corollary 86 (of Observation 73)**.** *For any u', any $(u', 1)$-maximin strategy of the game instance $\mathcal{B}(n, m, \mathbf{w}')$ with the updated weights is a $((1-\epsilon/2)u', 1)$-maximin strategy of the original instance $\mathcal{B}(n, m, \mathbf{w})$.*

Corollary 86 implies that to obtain a $((1-\epsilon)u, 1)$-maximin strategy on the original instance $\mathcal{B}(n, m, \mathbf{w})$, it suffices to find a $((1-\epsilon/2)u, 1)$-maximin strategy on instance $\mathcal{B}(n, m, \mathbf{w}')$. Therefore, from now on, we only focus on the instance with the updated weights and, for simplicity of notations, denote the updated weight of battlefield i by w_i.

Updating the battlefield weights in the aforementioned way results in reducing the total number of distinct weights down to $O(1/\epsilon \cdot \log u)$ (recall that the maximum weight among original weights was assumed to be at most u). We later show how this can be used to represent pure strategies of player 1 in a different way that

results in a significantly fewer number of strategies.

Step 2: Partitioning the battlefields into heavy and light subsets. We set a threshold $\tau = \epsilon u/2$ and partition the battlefields into two subsets of *heavy* battlefields with weights of at least τ and *light* battlefields with weights of less than τ. Roughly speaking, for light battlefields, we can afford to be the weight of one battlefield away from the optimal strategy and remain $(1 - \epsilon/2)$-competitive since the maximum weight among them is bounded by $\epsilon u/2$. For the heavy battlefields, however, we need to be more careful as even one battlefield might have a huge impact on the outcome of the game. The idea is to significantly reduce the number of strategies of the players that have to be considered on heavy battlefields so that we can consider them all. Let us denote by k_d the number of distinct battlefield weights; further, we denote by k_d^h the number of distinct battlefield weights that are larger than τ (i.e., are heavy). The following observation bounds k_d^h by a constant.

Observation 87. *The number of distinct heavy battlefield weights, k_d^h, is bounded by a constant.*

Proof. Recall that we assume no battlefield has weight more than u. Moreover, all heavy battlefields have weight at least $\epsilon u/2$. Since we updated the weight of all battlefields to be in set $\{1, (1+\epsilon/2)^1, (1+\epsilon/2)^2, \ldots\}$, it leaves only $\log_{1+\epsilon/2}(2/\epsilon) = \frac{\log(2/\epsilon)}{\log(1+\epsilon/2)} \leq 4/\epsilon \cdot \log(2/\epsilon)$ values in range $[\epsilon u/2, u]$ which is a constant number since ϵ is a constant. □

We show by Claim 88 that there always exists an optimal strategy that puts roughly the same number of troops in all battlefields of the same weight. Intuitively,

the players are indifferent to battlefields of the same weights and therefore the first player has no incentive to put significantly more troops on one of them.

Claim 88. *If player 1 has a* $(u, 1)$*-*maximin *strategy, he also has a* $(u, 1)$*-*maximin *strategy* $\mathbf{x}$ *where for any two battlefields* i *and* j *with* $w_i = w_j$*, we have* $|x_i - x_j| \leq 1$*.*

Proof. Consider two battlefields j and j' with the same weights and assume that for $(u, 1)$-maximin strategy $\hat{\mathbf{x}}$, we have $\hat{x}_j \geq \hat{x}_{j'} + 2$. We construct another strategy $\mathbf{x}'$ in the following way: $x'_j = \hat{x}_j - 1$, $x'_{j'} = \hat{x}_{j'} + 1$, and $x'_i = \hat{x}_i$ for all other battlefields. It is easy to confirm that $\mathbf{x}'$ is also a $(u, 1)$-maximin strategy. Applying the same function to $\mathbf{x}'$, we can inductively update the strategy to finally achieve a strategy $(u, 1)$-maximin strategy $\mathbf{x}$ for which we have $|x_i - x_j| \geq 1$ for all battlefields of the same weight. □

Now, the idea is to represent the pure strategies of player 1 differently. That is, instead of specifying the number of troops that are put on each battlefield, we represent each valid pure strategy of player 1 by the number of troops that are put on each battlefield weight. By Claim 88, this uniquely determines an optimal strategy without loss of generality. Therefore the dimension of each pure strategies of player 1 is now changed from k to $O(1/\epsilon \cdot \log u)$. Note that this is not sufficient to solve the problem in polynomial time since the number of pure strategies of player 1 can still be up to $n^{O(1/\epsilon \cdot \log u)}$. However, we show how this can bound the number of pure strategies of player 1 on heavy battlefields.

We define a *partial strategy* of player 1 on heavy battlefields to be a strategy in the new representation that only specifies how player 1 plays on heavy battlefields.

Formally, each partial strategy of player 1 on heavy battlefields can be represented by a vector $\mathbf{x}^h$ of length k_d^h, with non-negative entries in $[n]$ that sum up to at most n. Let us denote the set of all such strategies by $\mathcal{S}_1^h$. The following observation bounds the total number of strategies in $\mathcal{S}_1^h$.

Observation 89. $|\mathcal{S}_1^h| \leq n^{\mathsf{poly}(1/\epsilon)}$.

Proof. Since by Observation 87, $k_d^h \leq \mathsf{poly}(1/\epsilon)$, each strategy in $\mathcal{S}_1^h$ is a vector of length at most $\mathsf{poly}(1/\epsilon)$. This, combined with the fact that each entry is an integer between 0 and n, leads to the desired bound. □

A similar analysis can show that the strategies of player 2 can also be bounded by a polynomial on heavy battlefields; however, for technical reasons that we will elaborate on later, it is crucial to further bound the number of pure strategies of player 2 on heavy battlefields by a constant. The key idea here, is that to prevent player 1 from achieving a payoff of u, player 2 can only lose in at most $2/\epsilon$ heavy battlefields. Similar to that of player 1, we represent each partial strategy of player 2 on heavy battlefields by a vector of length k_d^h. However, instead of specifying the number of troops that player 2 puts on each battlefield weight, we only specify how many battlefields of each weight player 2 loses in. Therefore, formally, each partial strategy of player 2 on heavy battlefields can be represented by a vector of length k_d^h where each entry is a non-negative integer and all entries sum up to at most $2/\epsilon$. Denoting the set of all these partial strategies by $\mathcal{S}_2^h$, we can bound its size to be a constant.

Observation 90. $|\mathcal{S}_2^h| \leq O(1)$.

Proof. By Observation 87, the vectors in $\mathcal{S}_2^h$ have $k_d^h \leq \mathsf{poly}(1/\epsilon)$ dimensions and each entry is a number in $[0, 2/\epsilon]$ therefore there are at most $\mathsf{poly}(1/\epsilon)^{2/\epsilon+1}$ such vectors which is $O(1)$ since ϵ is a constant. □

Step 3: Solving the problem against a weaker adversary. Given a strategy $\mathbf{x}$ of player 1, the second player's best response is a strategy that maximizes his payoff. That is, player 2 seeks to find a strategy $\mathbf{y} \in \mathcal{S}_2$ such that $\mathsf{u}_2(\mathbf{x}, \mathbf{y})$, which is the weight of the battlefields in which $\mathbf{y}$ puts more troops than $\mathbf{x}$, is maximized. This is precisely equivalent to the following knapsack problem: The knapsack has capacity m (the number of troops of player 2) and for any $i \in [k]$, there exists an item with weight x_i and value w_i. Indeed one of the main challenges in finding an optimal $(u, 1)$-$\mathsf{maximin}$ strategy for player 1 is in that the second player's best response problem is a rather complicated function that makes it hard for the first player to optimize his strategy against. To resolve this issue, we optimize player 1's strategy against a *weaker adversary* than player 2. That is, we assume that the second player, instead of solving the aforementioned knapsack problem optimally, follows a simpler and more predictable algorithm. Note that there is a trade-off on the best-response algorithm that we fix for the weaker adversary. If it is too simplified, the strategy that we find for player 1 against it cannot perform well against player 2 who best-responds optimally; and if it is not simplified enough, there is no gain in considering the weaker adversary instead of player 2. To balance this, we allow the weak adversary to try all possible strategies on heavy battlefields, but only allow him to combine it with a greedy algorithm on light battlefields; this

process is formalized in Algorithm 10.[4] As we show in the rest of this section, while being only ϵ away from the optimal best-response algorithm, it becomes possible to compute an optimal strategy of player 1 against the weak adversary who responds by Algorithm 10 in polynomial time.

Algorithm 10: Weaker adversary's best response algorithm.

Input: a strategy $\mathbf{x} = (x_1, \ldots, x_k)$ of player 1.
Output: a strategy $\mathbf{y} = (y_1, \ldots, y_k)$ of player 2.
For every battlefield i, denote its ratio r_i to be w_i/x_i.
Let $\sigma(i)$ denote the index of the ith light battlefield with the highest ratio.
$\mathbf{y} \leftarrow (0, \ldots, 0)$
for every possible strategy $\mathbf{y}'$ of player 2 that puts non-zero troops only on heavy battlefields **do**
    $i \leftarrow 1$
    while $m - \sum_i y'_i \geq x_{\sigma(i)}$ and $i \leq k$ **do** ▷ Play greedily on light battlefields based on their ratios.
        $y'_{\sigma(i)} \leftarrow x_{\sigma(i)}$
        $i \leftarrow i + 1$
    end while
    if $\mathsf{u}_2(\mathbf{x}, \mathbf{y}') \geq \mathsf{u}_2(\mathbf{x}, \mathbf{y})$ **then** update $\mathbf{y} \leftarrow \mathbf{y}'$
end for
return $\mathbf{y}$

Lemma 91. *For any strategy* $\mathbf{x}$ *of player 1, let* $\mathbf{y}$ *be the optimal best-response of player 2 and let* $\tilde{\mathbf{y}}$ *be the strategy obtained by Algorithm 10. We have* $\mathsf{u}_2(\mathbf{x}, \tilde{\mathbf{y}}) \geq \mathsf{u}_2(\mathbf{x}, \mathbf{y}) - \frac{\epsilon u}{2}$.

Proof. Consider the iteration of the for loop in Algorithm 10 where the chosen partial strategy $\mathbf{y}'$ over the heavy battlefields is exactly equal to that of the optimal best-response $\mathbf{y}$ of player 2. We claim that after this strategy is combined with the

[4]For ease of exposition, we do not explicitly mention how to sort the battlefields when two have the same ratio in Algorithm 10. This can be simply handled by assuming that the battlefield with a lower index is preferred by the weaker adversary.

greedy algorithm over light battlefields to obtain a potential strategy $\mathbf{p}$, it provides a payoff of at least $\mathsf{u}_2(\mathbf{x}, \mathbf{y}) - \epsilon u/2$. Note that this is sufficient to prove the lemma since the returned solution $\mathbf{y}$ guarantees a payoff against $\mathbf{x}$ that is not less than that of $\mathbf{p}$.

Observe that we can safely ignore heavy battlefields since both $\mathbf{y}$ and $\mathbf{p}$ choose similar strategies over them. It only suffices to guarantee that the difference over light battlefields is at most $\epsilon u/2$. Let us denote by m' the number of troops that are left for player 2 over light battlefields. The optimal best-response is equivalent to the solution of the following knapsack problem: The knapsack has capacity m', and for each light battlefield i, there is an item with cost x_i (since player 2 has to put at least x_i troops to win it) and value w_i. Observe that the greedy approach of the weaker-adversary on light battlefields, is equivalent to a greedy algorithm for this knapsack problem where items are sorted based on their value per weight (i.e., w_i/x_i) and chosen greedily. This is in fact, the well-known greedy algorithm of knapsack that is known to guarantee an additive error of up to the maximum item value. Since here we only consider light battlefields and the maximum item value, or equivalently, the maximum battlefield weight is at most $\epsilon u/2$, we lose an additive error of at most $\epsilon u/2$ compared to optimal strategy $\mathbf{y}$, which concludes the proof. □

Corollary 92. *Any strategy* $\mathbf{x}$ *of player 1 that obtains a payoff of* u *against the weaker adversary, obtains a payoff of at least* $(1-\epsilon/2)u$ *against all possible strategies of player 2.*

Proof. Since the game is constant sum with $\sum_i w_i$ being sum of utilities, the weaker adversary gets a payoff of at most $\sum_i w_i - u$ against $\mathbf{x}$. Therefore since by Lemma 91 the optimal best response of player 2 obtains a payoff of at most $\sum_i w_i - u + \epsilon u/2$. This means that the first player gets a payoff of at least $u - \epsilon u/2 = (1 - \epsilon/2)u$ against *any* strategy of the second player. □

By Corollary 92, to obtain the desired $((1 - \epsilon/2)u, 1)$-maximin strategy, it suffices to guarantee a payoff of u against the weaker adversary. Note that such strategy is guaranteed to exist since existence of a $(u, 1)$-maximin strategy against player 2 is guaranteed.

A wrong approach. We start with a wrong approach in guaranteeing a payoff of u against the weaker adversary and later show how to fix its shortcomings. Observe that the weaker adversary exhaustively searches through all of his possible strategies among heavy battlefields and plays greedily on light battlefields. Therefore, the first player needs to approach heavy and light battlefields differently. One way of doing this is to fix the number of troops that each of the players will spend on heavy and light battlefields, which is bounded by $O(nm)$, and solve two instances independently. On light battlefields, since player 2 responds greedily, it is not hard to optimize the first player's strategy. On heavy battlefields, the number of pairs of valid strategies of the players are bounded by a polynomial by Observations 110 and 111. Therefore, player 1 can find his optimal strategy among them by exhaustively checking all of his strategies against all possible responses of player 2.

The problem with this approach, is that it is not possible to solve the sub-

problems on heavy and light battlefields independently. More precisely, the number of troops that player 2 puts on heavy and light battlefields is a direct function of the strategy of player 1. This implies that it is not possible to fix the number of troops of player 2 over heavy and light battlefields a priori, without specifying the exact strategy of player 1.

The correct approach. To resolve the aforementioned problem, we do not fix the budget of player 2 on light and heavy battlefields beforehand. Instead, we first only fix the strategy $\mathbf{x}^h \in \mathcal{S}_1^h$ of player 1 on heavy battlefields. (By Observation 110 there are only polynomially many such strategies, therefore it is possible to try them all.) Next, note that player 2, by Observation 111, has only a constant number of responses to $\mathbf{x}^h$ that we would have to consider (other responses simply guarantee us a payoff of u on heavy battlefields alone). Roughly speaking, while finding the optimal strategy of player 1 on light battlefields, we would have to consider all these responses of player 2 on heavy battlefields. That is, each response $\mathbf{y}^h \in \mathcal{S}_2^h$ of player 2 on heavy battlefields uniquely determines (1) what payoff the players get on heavy battlefields, (2) how many troops is left for player 2 to play on light battlefields. The strategy that we find for player 1 on light battlefields, has to perform well against all of these responses.

In a thought experiment, consider the optimal strategy $\mathbf{x}^\ell$ of player 1 on light battlefields, given that his strategy on heavy battlefields is fixed to be $\mathbf{x}^h$. For every strategy $\mathbf{y}^h$ of player 2 on heavy battlefields, the greedy algorithm of player 2 on light battlefields first sorts light battlefields based on their ratios w_i/x_i^ℓ and greedily

wins battlefields of higher ratio. This means that for every response of player 2 on heavy battlefields, there is a unique light battlefield of highest ratio that player 2 loses in. Let us denote by b_i the index of this battlefield for the ith response of player 2 in $\mathcal{S}_2^h$. We also denote by $r_i := w_{b_i}/x_{b_i}^\ell$ the ratio of battlefield b_i based on $\mathbf{x}^\ell$. The takeaway, here, is that given the optimal strategy $\mathbf{x}^\ell$ of player 2 on light battlefields, we can uniquely determine vectors $\mathbf{b} = (b_1, b_2, \ldots, b_{|\mathcal{S}_2^h|})$ and $\mathbf{r} = (r_1, r_2, \ldots, r_{|\mathcal{S}_2^h|})$. We have no way of knowing these vectors without knowing $\mathbf{x}^\ell$, however, we find the right value of them by checking all possible vectors. To achieve this, the first step is to show that there are only polynomially many options that we need to try.

Claim 93. *There are only polynomially many valid triplets* $(\mathbf{x}^h, \mathbf{b}, \mathbf{r})$.

Proof. We already know from Observation 110 that there are only polynomially many possible choices for $\mathbf{x}^h$. To conclude the proof, it suffices to show that for the choice of $\mathbf{b}$ and $\mathbf{r}$, we also have polynomially many possibilities. To see this, note that $\mathbf{b}$ and $\mathbf{r}$ only have $|\mathcal{S}_2^h|$ dimensions which is bounded by a constant by Observation 111. Furthermore, each entry of $\mathbf{b}$ is a number in $\{0, 1, \ldots, k\}$ and each entry of $\mathbf{r}$ has at most $O(nk)$ values. This means there are only $k^{O(1)}$ many possibilities for $\mathbf{b}$ and $(nk)^{O(1)}$ many possibilities for $\mathbf{r}$. Both of these upper bounds are polynomial, which proves there are only polynomially many triplets $(\mathbf{x}^h, \mathbf{b}, \mathbf{r})$.

□

The next step is to find a strategy $\mathbf{x}^\ell$ of player 1 on light battlefields that *satisfies* a given triplet $(\mathbf{x^h}, \mathbf{b}, \mathbf{r})$. We start by formalizing what satisfying exactly means.

Definition 94. *We say a strategy $\mathbf{x}^\ell$ of player 1 on light battlefields satisfies a triplet $(\mathbf{x}^h, \mathbf{b}, \mathbf{r})$ if the following conditions hold.*

1. *The number of troops that are used in $\mathbf{x}^\ell$ is at most $n - \sum_i x_i^h$.*
2. *If player 2 chooses his ith strategy from $\mathcal{S}_2^h$ over heavy battlefields and plays greedily (according to Algorithm 10) on light battlefields, the highest ratio light battlefield in which he loses becomes b_i.*
3. *The ratio $w_{b_i}/x_{b_i}^\ell$ is indeed equal to r_i.*

Note that it is not necessarily possible to satisfy any given triplet $(\mathbf{x^h}, \mathbf{b}, \mathbf{r})$. However, it is guaranteed that the right choice of it, where $\mathbf{x^h}$ corresponds to the optimal partial strategy of player 1 on heavy battlefields and $\mathbf{b}$ and $\mathbf{r}$ correspond to their actual value (as described before) for the optimal strategy of player 1, is satisfiable. Recall that our goal is to exhaustively try all possible triplets and find the optimal one. To achieve this, we need an oracle that confirms whether a given triplet is satisfiable. Lemma 95 provides this oracle via a dynamic program that further guarantees that the obtained strategy is maximin against the weak adversary given that player 1 is committed to satisfy $(\mathbf{x}^h, \mathbf{b}, \mathbf{r})$.

Lemma 95. *Given a triplet $\mathcal{T} = (\mathbf{x}^h, \mathbf{b}, \mathbf{r})$, one can in polynomial time, either report that $\mathcal{T}$ is not satisfiable, or find a strategy $\mathbf{x}^\ell$ that satisfies $\mathcal{T}$ while guaranteeing that the payoff of the response of the weaker adversary to strategy $(\mathbf{x}^h, \mathbf{x}^\ell)$ of player 1 that is obtained by Algorithm 10 is minimized.*

Proof. For simplicity of notations, let us denote by $c := |\mathcal{S}_2^h|$ the total number of responses of player 2 over the heavy battlefields, and consequently, the size of vectors

b and **r**. Furthermore, given that player 1 plays strategy $\mathbf{x}^h$ on heavy battlefields and that player 2 plays his ith response in $\mathcal{S}_2^h$ to $\mathbf{x}^h$, we denote by m_i the number of troops that are left for player 2 to play on light battlefields, and denote respectively by $g_1(i)$ and $g_2(i)$ the guaranteed payoff that players 1 and 2 get on heavy battlefields.

To satisfy condition 1 of Definition 94, we ensure that the strategy that we find for player 1 over the light battlefields only uses $n^\ell := n - \sum_i x_i^h$ troops. Satisfying the third condition is also straightforward: for any $i \in [c]$, it suffices for player 1 to put exactly w_{b_i}/r_i troops on battlefield b_i. (We emphasize that we should not change the number of troops on these battlefields throughout the algorithm.) If during this process, we need to use more than n^ℓ troops or if for some i and j we have $b_i = b_j$ and $w_{b_i}/r_i \neq w_{b_j}/r_j$ we report that the given triplet is not satisfiable. The main difficulty is to ensure that the second condition of Definition 94 also holds. For that, the only decision that we have to make is on the number of troops that we put on the remaining battlefields. For our final strategy $\mathbf{x}^\ell$ over the light battlefields, assume that the battlefields are sorted decreasingly based on their ratio w_i/x_i^ℓ and let us denote by $\sigma(i)$ the index of the battlefield in position i. To convey the overall idea of how to satisfy this condition, let us first focus on the ith strategy of player 2 over the heavy battlefields. We need to make sure that in our strategy $\mathbf{x}^\ell$ over light battlefields, the ratio of battlefields are such that if player 2 wins them greedily, he stops at battlefield b_i. More precisely, let $\sigma(\gamma) = b_i$, we need to have[5]

$$0 \leq m_i - \sum_{j=1}^{\gamma-1} x_{\sigma(j)}^\ell < x_{b_i}^\ell. \tag{5.14}$$

[5]In case of a tie, we assume the battlefield with the lower index has a higher ratio.

Fix battlefield b_i, roughly speaking, in order to satisfy (5.14), we need to decide which battlefields will have a higher ratio than battlefield b_i so as to ensure that once player 2 wins them, he will have less than $x^{\ell}_{b_i}$ troops. Now recall that we need to satisfy this, simultaneously, for every entry b_i of $\mathbf{b}$. For this we use a dynamic program $D(j, n', \omega_1, \ldots, \omega_c, u_1, \ldots, u_c)$ with $j \in \{0, \ldots, k^{\ell}\}$, $n' \in \{0, , \ldots, n^{\ell}\}$, $u_i \in \{0, \ldots, \sum_i w_i\}$, and $\omega_i \in \{0, 1, \ldots, m_i\}$. The value of $D(j, n', \omega_1, \ldots, \omega_c, u_1, \ldots, u_c)$ is either 0 or 1 and it is 1 iff it is possible to give a partial strategy $(x_1, \ldots, x_j)$ over the first j light battlefields such that all the following conditions are satisfied.

1. We use exactly n' troops over them, i.e.,

$$\sum_{j' \in [j]} x_{j'} = n'. \tag{5.15}$$

2. For any $i \in [c]$, we put at least ω_i troops in battlefields with ratio higher than r_i, i.e.,

$$\sum_{j' \in [j] : \frac{w_{j'}}{x_{j'}} \geq r_i} x_{j'} \geq \omega_i, \qquad \forall i \in [c]. \tag{5.16}$$

3. For any $i \in [c]$, sum of weights of battlefields with lower ratio than r_i is at least $u_i - g_1(i)$, i.e.,

$$\sum_{j' \in [j] : \frac{w_{j'}}{x_{j'}} < r_i} w_{j'} \geq u_i - g_1(i), \qquad \forall i \in [c]. \tag{5.17}$$

It is easy to confirm, by definition, that we can satisfy $(\mathbf{x}^h, \mathbf{b}, \mathbf{r})$ iff,

$$D(k^\ell, n^\ell, m_1 - x^\ell_{b_1} - 1, \ldots, m_c - x^\ell_{b_c} - 1, 0, \ldots, 0) = 1.$$

To further maximize the payoff that player 1 gets against the weaker adversary as well as satisfying the given triplet it suffices to find the maximum value of u where

$$D(k^\ell, n^\ell, m_1 - x^\ell_{b_1} - 1, \ldots, m_c - x^\ell_{b_c} - 1, u, \ldots, u) = 1.$$

Base case. We start with the base case, where $j = 0$. Here, clearly, if $\omega_i > 0$ or $u_i > 0$ for some i, then the value of D must be 0 as we have no way of satisfying (5.16) or (5.17). Moreover, if $n' > 0$, the value must again be 0, as (5.15) requires that we need to spend exactly n' troops over the first j battlefields. Therefore, the only case where the value of the case where $j = 0$ is one is where $n' = 0$, and $\omega_i = 0$ and $u_i = 0$ for all $i \in [c]$.

Updating the DP. To update $D(j, n', \omega_1, \ldots, \omega_c, u_1, \ldots, u_c)$, we only have to decide on how many troops to put on the jth battlefield. The idea is to try all possibilities and check, recursively, whether any of these choices satisfies the requirements for thy dynamic value to be 1. Let us denote by $P = \{0, \ldots, n'\}$ the set of all possibilities for the number of troops that we can put on battlefield j. We update D as follows:

$$D(j, n', \omega_1, \ldots, \omega_c, u_1, \ldots, u_c) = \max_{x \in P} D\Big(j-1, n'-x, \omega_1(x, j), \ldots, \omega_c(x, j), u_1(x, j), \ldots, u_c(x, j)\Big)$$

where for any $i \in [c]$, $\omega_i(x, j)$ and $u_i(x, j)$ are defined as

$$\omega_i(x,j) = \begin{cases} \omega_i, & \text{if } w_j/x < r_d, \\ \omega_i - x, & \text{otherwise,} \end{cases} \quad \text{and,} \quad u_i(x,j) = \begin{cases} u_i - w_j, & \text{if } w_j/x < r_d, \\ u_i, & \text{otherwise.} \end{cases}$$

Recall that when $j = b_i$ for some $i \in [c]$, as previously mentioned, to prevent violation of condition 3 of Definition 94, we have to put exactly w_j/r_i troops on battlefield j. If this is the case, and we cannot afford w_j/r_i troops (i.e., if $w_j/r_i > n'$), we update D to be 0. If $w_j/r_i \leq n'$, we make an exception for battlefield j and overload the set P to be $\{w_j/r_i\}$ instead of $\{0, \ldots, n'\}$; the rest of the updating procedure would be the same.

Correctness. Here, we argue that our updating procedure produces the right answer. We use induction on the value of j and assume that $D(j', n', \omega_1, \ldots, \omega_c, u_1, \ldots, u_c)$ is correctly updated for $j' < j$. We argued why the base case, where $j = 0$ is correctly updated. It only suffices to prove for that we correctly update $D(j, n', \omega_1, \ldots, \omega_c, u_1, \ldots, u_c)$. Assume for now that $D(j, n', \omega_1, \ldots, \omega_c, u_1, \ldots, u_c) = 1$. This implies, by definition, that there exists a strategy $(x_1, \ldots, x_j)$ over the first j light battlefields that satisfies (5.15), (5.16), and (5.17). Observe that $0 \leq x_j \leq n'$, and thus, $x_j \in P$. Once putting x_j troops in battlefield j, $\omega_i(x_j, j)$ denotes the number of troops that still has to be put on battlefields with higher ratio than r_i. Moreover, $u_i(x_j, j)$ denotes the requirement for the sum of weights of battlefields with lower ratio than r_i. Therefore, by (5.16), and (5.17) we need to have $D\big(j-1, n'-x_j, \omega_1(x_j, j), \ldots, \omega_c(x_j, j), u_1(x_j, j), \ldots, u_c(x_j, j)\big) = 1$ concluding this case. On the

other hand, if $D(j, n', \omega_1, \ldots, \omega_c, u_1, \ldots, u_c) = 0$, no such strategy $(x_1, \ldots, x_j)$ can be found, therefore for all choices of x_j in P, the requirements over the prior battlefields cannot be satisfied and $D\big(j-1, n'-x_j, \omega_1(x_j, j), \ldots, \omega_c(x_j, j), u_1(x_j, j), \ldots, u_c(x_j, j)\big) = 0$ for all choices of $x_j \in P$.

Wrap up. To obtain the strategy $(\mathbf{x}^h, \mathbf{x}^\ell)$ of player 1 that satisfies the triplet $(\mathbf{x}^h, \mathbf{b}, \mathbf{r})$ while providing the maximum guaranteed payoff against all possible strategies of the weaker adversary, we solve the aforementioned dynamic program and linear search over $[0, \sum_i w_i]$ to find the maximum value of u where

$$D(k^\ell, n^\ell, m_1 - x^\ell_{b_1} - 1, \ldots, m_c - x^\ell_{b_c} - 1, u, \ldots, u) = 1.$$

While this only outputs the maximum payoff that can be guaranteed — and not the actual strategy to provide it — it is easy to construct the strategy using standard DP techniques. Roughly speaking, to achieve this, we need to slightly modify the DP to also store the actual partial strategy in case its value is 1. □

To conclude this section, for any choice of the triplet $\mathcal{T} = (\mathbf{x}^h, \mathbf{b}, \mathbf{r})$ for which there are only polynomially many options by Claim 93, we find the optimal strategy of player 1 among light battlefields that satisfies $\mathcal{T}$ if possible. Lemma 95 guarantees that the obtained strategy over the light battlefields is indeed the optimal solution if the choice of triplet $\mathcal{T}$ is right. Therefore, it suffices to compare the guaranteed payoff of all obtained solutions for different triplets and report the one that provides the maximum guaranteed payoff for player 1. This procedure gives us the optimal

strategy of player 1 against the weaker adversary and therefore it guarantees a payoff of u against him. By Corollary 92, this gives a $((1-\epsilon/2)u, 1)$-maximin strategy against player 2. Recall that this is achieved over the updated battlefield weights. However, by Corollary 86, any $((1-\epsilon/2)u, 1)$-maximin strategy of player 1 on updated battlefields, gives a $((1-\epsilon)u, 1)$-maximin on the original game instance and this proves the main theorem of this section.

Theorem 85 (restated). *Given that player 1 has a $(u, 1)$-maximin strategy, there exists a polynomial time algorithm that obtains a $((1-\epsilon)u, 1)$-maximin strategy of player 1 for any arbitrarily small constant $\epsilon > 0$.*

5.4.2 The Case of 2-Strategies

In this section we generalize the results of the previous section to the case of 2-strategies. To achieve this we first design an algorithm that obtains a $(u/3, p)$-maximin 2-strategy. Then, we use this algorithm and adapt some of the techniques from Section 5.4.1 to give an algorithm that finds a $((1-\epsilon)u, p)$-maximin 2-strategy.

The proof of the following theorem which is the main result of this section comes later in the section.

Theorem 96. *Given that player 1 has a (u, p)-maximin 2-strategy, there exists a polynomial time algorithm that obtains a $((1-\epsilon)u, p)$-maximin 2-strategy of player 1 for any arbitrarily small constant $\epsilon > 0$.*

Given that the first player randomizes over two pure strategies $\mathbf{x}$ and $\mathbf{x'}$, the second player's best response is a strategy that maximizes his payoff. That is, player

2 seeks to find a strategy $\mathbf{y} \in \mathcal{S}_2$ such that $\min(\mathsf{u}_2(\mathbf{x}, \mathbf{y}), \mathsf{u}_2(\mathbf{x'}, \mathbf{y}))$ is maximized. In this case, we assume that player 1 cannot guarantee utility u for himself, otherwise he would just play one pure strategy. Therefore, by Observation 78 and Observation 79 We can safely assume that in any (u, p)-maximin 2-strategy of the first player p=1/2 holds.

Although, there exist polynomial time algorithms to find the best response of player 2, we still need simpler algorithms to be able to use them and find an optimal $((1-\epsilon)u, p)$-maximin 2-strategy for player 1. To this end, we define a new opponent which is a weaker version of the second player and we call it the *greedy opponent*. Instead of solving the best response problem optimally, the greedy opponent just takes a simple greedy approach. We prove that playing against this weaker opponent gives us a $(u/3, p)$-maximin 2-strategy. To approach that we first give an alternative formulation of the best response problem in which the solution is represented by two binary vectors. We then relax the integrality condition of elements in the vectors and design a greedy algorithm that finds the best response of the second player in this case. This algorithm is the base for the greedy opponent's strategy. We prove that if the second player takes this approach he loses at most $2 \cdot \mathsf{w}_{\mathsf{max}}$ payoff compared to his best response strategy. Recall that $\mathsf{w}_{\mathsf{max}}$ is the maximum weight of the battlefields.

To give an alternative formulation of the best response problem, we define a new problem in which any solution is represent by two vectors of size k. Given strategies $\mathbf{x}$ and $\mathbf{x'}$ we first define *cost* vectors $\mathbf{c}$ and $\mathbf{c'}$. For any battlefield i, if $x_i \leq x'_i$ then $c_i := x_i$ and $c'_i := x'_i - x_i$. Otherwise, $c'_i := x'_i$ and $c_i := x_i - x'_i$.

Roughly speaking, when $x_i \leq x'_i$ holds, c_i is the number of troops that player 2 needs to put in battlefield i to win strategy $\mathbf{x}$ in this battlefield and c'_i is the number of troops that he should add to win both strategies. The solution to this problem is two vectors $\mathbf{h}$ and $\mathbf{h'}$ that maximizes $\min(\mathbf{w} \cdot \mathbf{h}, \mathbf{w} \cdot \mathbf{h'})$ subject to the following conditions.

1. For any $i \in [k]$, elements h_i and h'_i are binary variables.

2. If $x_i \leq x'_i$, then $h'_i \leq h_i$ holds. Otherwise, $h_i \leq h'_i$ holds.

3. Also, $\mathbf{c} \cdot \mathbf{h} + \mathbf{c'} \cdot \mathbf{h'} \leq m$.

Given vectors $\mathbf{h}$ and $\mathbf{h'}$, the solution of this problem, one can find a strategy $\mathbf{y}$ of player 2 where $u = \mathbf{w} \cdot \mathbf{h}$ and $u' = \mathbf{w} \cdot \mathbf{h}$. (Recall that u and u' respectively denote the utility that strategy $\mathbf{y}$ gets against strategies $\mathbf{x}$ and $\mathbf{x'}$.) We claim that if for any $i \in [k]$, player 2 puts $h_i \cdot c_i + h'_i \cdot c'_i$ troops in battlefield i, then $u \geq \mathbf{w} \cdot \mathbf{h}$ and $u' \geq \mathbf{w} \cdot \mathbf{h}$ hold. Note that by condition 3, he has enough troops to play this strategy. Without loss of generality, assume that $x_i \leq x'_i$. Therefore, by condition 2, $h'_i \leq h_i$ holds. There are three possible cases. Either both h'_i and h_i are 0, both are equal to 1, or $h'_i = 0$ and $h_i = 1$. If $h_i = h'_i = 0$, then $w_i \cdot h_i$ and $w'_i \cdot h'_i$ are both equal to zero. In the case of $h_i = 1$ and $h_i = 0$, we have $h_i \cdot c_i + h'_i \cdot c'_i = x_i$. Therefore, in battlefield i, player 2 gets utility w_i against strategy x_i. Also, if $h_i = h'_i = 1$, the equality $h_i \cdot c_i + h'_i \cdot c'_i = x'_i$ holds which means player 2 gets utility w_i against both strategies in this battlefield. Thus, in all three cases, the utility that player 2 gets against strategies $\mathbf{x}$ and $\mathbf{x'}$ in battlefield i is receptively at least $w_i \cdot h_i$ and at least $w'_i \cdot h'_i$.

Moreover, given any strategy $\mathbf{y}$ of player 2, one can give two valid vectors $\mathbf{h}$ and $\mathbf{h'}$ such that $u = \mathbf{w} \cdot \mathbf{h}$ and $u' = \mathbf{w} \cdot \mathbf{h}$. Iff $y_i \geq x_i$, let $h_i := 1$. Also iff $y_i \geq x'_i$, we set $h'_i := 1$. Therefore, $u = \mathbf{w} \cdot \mathbf{h}$ and $u' = \mathbf{w} \cdot \mathbf{h}$ hold. We also need to show that $\mathbf{c} \cdot \mathbf{h} + \mathbf{c'} \cdot \mathbf{h'} \leq m$. For this, it suffices to show $c_i \cdot h_i + c'_i \cdot h'_i \leq y_i$ for any $i \in [k]$. Without loss of generality, assume that $x_i \leq x'_i$. Thus, $c_i = x_i$ and $c'_i = x'_i - x_i$ hold. Note that we can assume that y_i is either 0, x_i or x'_i. In all these three cases, $c_i \cdot h_i + c'_i \cdot h'_i \leq y_i$ holds; therefore, vectors $\mathbf{h}$ and $\mathbf{h'}$ satisfy the necessary conditions.

To sum up, for any solution of the defined problem which we denote by vectors $\mathbf{h}$ and $\mathbf{h'}$ there is a strategy of the opponent with payoff $\mathbf{c} \cdot \mathbf{h} + \mathbf{c'} \cdot \mathbf{h'}$. Also, for any (u, p)-maximin 2-strategy of player 2, there are two vectors $\mathbf{h}$ and $\mathbf{h'}$ that satisfy the necessary conditions of the problem and $\min(\mathbf{w} \cdot \mathbf{h}, \mathbf{w} \cdot \mathbf{h'}) = u$ holds for them. Therefore, this formulation is indeed a valid formulation of the best response problem. From now on, we may use these two formulations interchangeably.

5.4.2.1 A 1/3-Approximation

Any strategy of the player 2 against a 2-strategy of the first player can be represented by two vectors $\mathbf{h}$ and $\mathbf{h'}$. If we relax the integrality constraint of the elements in these vectors, they can be fractional numbers between 0 and 1. We call such a strategy a fractional strategy. In this section, we give an algorithm to find the best fractional strategy of the opponent. Using this algorithm we give an exact definition of the greedy opponent. Also, using some properties of the algorithm we prove that player 1 achieves a $(u/3, p)$-maximin strategy by finding his best strategy

against the greedy opponent.

Given $s = (\mathbf{x}, \mathbf{x'})$ a 2-strategy of the first player, vectors $\mathbf{h}$ and $\mathbf{h'}$ are a valid representation of a fractional response $\mathbf{y}$ iff:

1. For any $i \in [k]$, h_i and h'_i are real numbers between 0 and 1.

2. If $x_i \leq x'_i$, then $h'_i \leq h_i$ holds. Otherwise, $h_i \leq h'_i$ holds.

3. Also, $\mathbf{c} \cdot \mathbf{h} + \mathbf{c'} \cdot \mathbf{h'} \leq m$.

The utility that this strategy gets against $\mathbf{x}$ and $\mathbf{x'}$ is respectively $\mathsf{u}_2(x, y) = \mathbf{w} \cdot \mathbf{h}$ and $\mathsf{u}_2(x', y) = \mathbf{w} \cdot \mathbf{h'}$. Let $(\mathbf{h}, \mathbf{h'})$ be such a strategy. We call an element h_i (or h'_i) an *available element* if it is possible to increase h_i by a nonzero amount without changing h'_i. Formally element h_i is available iff the following conditions hold: $h_i < 1$ and if $x'_i \leq x_i$, then $h_i \leq h'_i$ holds. We also call h_i and h'_i *jointly available* iff $h_i < 1$ and $h'_i < 1$. In the other words, it is possible to increase both of them by a nonzero amount without violating the necessary conditions on $\mathbf{h}$ and $\mathbf{h'}$. Let $\mathbf{c}$ and $\mathbf{c'}$ be the cost vectors of a 2-strategy of the first player denoted by s. We define two *ratio vectors* of length k for this strategy and denote them by $\mathbf{r}$ and $\mathbf{r'}$. For any battlefield $i \in [k]$, we set $r_i = \frac{c_i}{w_i}$ and $r'_i = \frac{c'_i}{w_i}$. Note that spending ϵ amount of troop on an available element, h_i, increases it by ϵ/c_i and increases $\mathsf{u}_2(x, y)$ by ϵ/r_i. Roughly speaking, elements with smaller ratios are more valuable.

Given a 2-strategy of the first player in an instance of the Colonel Blotto, Algorithm 11 find a best fractional response of the second player. This algorithm consists of several iterations. At the beginning, all the elements of vectors $\mathbf{h}$ and $\mathbf{h'}$ are 0. In each iteration, the algorithm chooses two elements h_i and h'_j where they

are independently or jointly available and they minimize $r_i + r'_j$. Then, it increases each one by a nonzero amount such that $\mathbf{h} \cdot \mathbf{w}$ and $\mathbf{h'} \cdot \mathbf{w}$ increase equally. Note that to increase these elements the algorithm spends an amount of troop that is determined by the cost vectors. This algorithm stops if there is no troop left or it is not possible to increase the elements. Note that the output of this algorithm satisfies equation $\mathbf{h} \cdot \mathbf{w} = \mathbf{h'} \cdot \mathbf{w}$. In the other words, the strategy that this algorithm finds gets the same amount of utility against strategies $\mathbf{x}$ and $\mathbf{x'}$. In Lemma 97, we prove that any best fractional response of the second player has this property. Also, we prove in Lemma 98 that this algorithm indeed finds a best fractional response.

Lemma 97. *If vectors* $\mathbf{h}$ *and* $\mathbf{h'}$ *represent a best fractional strategy of the player 2 against an arbitrary 2-strategy of the first player, then* $\mathbf{w} \cdot \mathbf{h} = \mathbf{w} \cdot \mathbf{h'}$ *holds.*

Proof. Let $\mathbf{y}$ be a best fractional strategy of the second player represented by $\mathbf{h}$ and $\mathbf{h'}$ where $\mathbf{w} \cdot \mathbf{h} \neq \mathbf{w} \cdot \mathbf{h'}$. Without loss of generality, assume $\mathbf{w} \cdot \mathbf{h} > \mathbf{w} \cdot \mathbf{h'}$. We show that it is possible to increase $\min(\mathbf{w} \cdot \mathbf{h}, \mathbf{w} \cdot \mathbf{h'})$ by modifying $\mathbf{y}$. Since $\mathbf{w} \cdot \mathbf{h} > \mathbf{w} \cdot \mathbf{h'}$ holds, there exists a battlefield $i \in [k]$ where $h_i > h'_i$. We can decrease h_i by a nonzero amount and increase h'_i using the extra amount of troop achieved from that. This modification of h_i and h'_i increases $\min(\mathbf{w} \cdot \mathbf{h}, \mathbf{w} \cdot \mathbf{h'})$ by a nonzero amount which is a contradiction with the assumption that $\mathbf{y}$ is a best fractional strategy. □

Lemma 98. *Algorithm 11 gives a best fractional response of the second player.*

Proof. Let strategy $\mathbf{y}$ denote the output of Algorithm 11. Assume $\mathbf{y}$ is not a best response of the second player and let strategy $\mathbf{y'}$ represented by a pair of vectors $\boldsymbol{\eta}$

Algorithm 11: Best fractional strategy of the second player.

Input: strategies $\mathbf{x}$ and $\mathbf{x'}$ of the first player and vector $\mathbf{w}$ which is the battlefields' weight vector.

Output: two vectors $\mathbf{h}$ and $\mathbf{h'}$ which denote a fractional strategy of player 2.

Let $\mathbf{c}, \mathbf{c'}$ denote cost vectors of $\mathbf{x}$ and $\mathbf{x'}$.
$\mathbf{h}, \mathbf{h'} \leftarrow (0, \ldots, 0)$
while $m' > 0$ **do**
    $B_1 \leftarrow \{i : i \in [k], h_i \text{ is available}\}$ and $a \leftarrow \text{argmin}_{i \in B_1}\ r_i$
    $B_1' \leftarrow \{i : i \in [k], h_i' \text{ is available}\}$ and $b \leftarrow \text{argmin}_{i \in B_1'}\ r_i'$
    $B_2 \leftarrow \{i : i \in [k], h_i \text{ and } h_i' \text{ are jointly available}\}$ and $d \leftarrow \text{argmin}_{i \in B_2}\ r_i + r_i'$

    ▷ *In the case of tie, pick the smaller index*

    if $r_a + r_b' < r_d + r_d'$ **then**
        Maximize $t + t'$ subject to the following conditions:
        - After increasing h_a by t and h_b' by t', vectors $\mathbf{h}$ and $\mathbf{h'}$ are valid.
        - $t \cdot w_a = t' \cdot w_b$
        - $t \cdot c_a + t' \cdot c_b' \leq m'$

        Increase h_a by t and h_b' by t'.
        $m' \leftarrow m' - (t \cdot c_a + t' \cdot c_b')$
    else
        Maximize t subject to the following conditions:
        - After increasing h_d and h_d' by t, vectors $\mathbf{h}$ and $\mathbf{h'}$ are valid.
        - $t \cdot (c_d + c_d') \leq m'$

        Increase both h_d and h_d' by t.
        $m' \leftarrow t \cdot (c_d + c_d')$
    end if
end while
return $\mathbf{h}, \mathbf{h'}$

and $\boldsymbol{\eta'}$ be a best response of player 2 against the 2-strategy $(\mathbf{x}, \mathbf{x'})$. Let t denote the first iteration of the algorithm after which there exists at least a battlefield $i \in [k]$ where $\eta_i < h_i$ or $\eta_i' < h_i'$ (Vectors $\mathbf{h}$ and $\mathbf{h'}$ are defined in Algorithm 11.) Let i and j denote the battlefields where h_i and h_j' are increased in the t-th iteration of the algorithm. (It is possible that $j = i$.) Assume that among all the best strategies of the second player $\mathbf{y'}$ is the strategy that minimizes t. Also among all those with minimum t, $\mathbf{y'}$ is the one that minimizes $(h_i - \eta_i) + (h_j' - \eta_j')$. We show that given

such a strategy we can modify it in a way that t or $(h_i - \eta_i) + (h'_i - \eta'_i)$ decreases which is a contradiction. Note that $\mathsf{u}_2(\mathbf{x}, \mathbf{y}) = \mathsf{u}_2(\mathbf{x'}, \mathbf{y})$ holds since each iteration of the Algorithm 11 increases them by the same amount. Therefore, strategy $\mathbf{y'}$ should perform better than $\mathbf{y}$ against both $\mathbf{x}$ and $\mathbf{x'}$. More formally, $\mathsf{u}_2(\mathbf{x}, \mathbf{y}) < \mathsf{u}_2(\mathbf{x}, \mathbf{y'})$ and $\mathsf{u}_2(\mathbf{x'}, \mathbf{y}) < \mathsf{u}_2(\mathbf{x'}, \mathbf{y'})$ hold. There are two possible cases for the relation between $\mathbf{h}$ and $\mathbf{h'}$ after iteration t. In both cases we prove that it is possible to modify $\mathbf{y'}$ and lower $(h_i - \eta_i) + (h'_i - \eta'_i)$ or t while it is still a best strategy of the second player.

Case 1: Both $h_i > \eta_i$ and $h'_j > \eta'_j$ hold. In this case, strategy $\mathbf{y}$ achieves more utility against $\mathbf{x}$ in battlefield i and against $\mathbf{x'}$ in battlefield j than strategy $\mathbf{y'}$ does. Therefore, strategy $\mathbf{y'}$ compensates this by putting more troops in other battlefields. If there is a battlefield $i' \in [k]$, where $h_{i'} < \eta_{i'}$ and $h'_{i'} < \eta'_{i'}$ then it is possible to modify strategy $\mathbf{y'}$ by decreasing $\eta_{i'}$ and $\eta'_{i'}$ by a nonzero amount and increasing η_i and η'_j such that the amount of troops used by strategy $\mathbf{y'}$ does not increase and its utility does not decrease. It is possible since Algorithm 11 in each iteration, among all the available elements chooses a pair that spending a unit of troop in them gives us the maximum amount of utility, and elements $h_{i'}$ and $h'_{i'}$ are both available at this iteration. If this does not hold then there are two battlefields $i', j' \in [k]$ where $h_{i'} < \eta_{i'}$ and $h'_{j'} < \eta'_{j'}$. In this case, the only difference is that we need to make sure that both $h_{i'}$ and $h'_{j'}$ are available at this iteration which can be inferred from the fact that for both i' and j', equations $h'_{i'} \geq \eta'_{i'}$ and $h_{j'} \geq \eta_{j'}$ hold.

Case 2: Exactly one of $h_i > \eta_i$ and $h'_j > \eta'_j$ holds. Without loss of generality, we assume $h_i > \eta_i$ holds. Note that by Lemma 97 the amount of utility that strategy $\mathbf{y'}$

gets against strategies $\mathbf{x}$ and $\mathbf{x'}$ is equal. Also, since $h_i > \eta_i$ and $h'_j \leq \eta'_j$ hold, there exists a battlefield $i' \in [k]$ where $\eta_{i'} - h_{i'} > \eta'_{i'} - h'_{i'}$. This means that $h_{i'}$ is available at iteration t. However, in each iteration, Algorithm 11, chooses a pair of elements in $\mathbf{h}$ and $\mathbf{h'}$ where the amount of utility achieved per spending a unit of troop in their combination is maximized. This yields that it is possible to modify $\mathbf{y'}$ by decreasing $\eta_{i'}$ and increasing η_i in a way that $\boldsymbol{\eta'}$ and $\boldsymbol{\eta}$ are still a valid representation of strategy $\mathbf{y'}$ and $\mathsf{u}_2(\mathbf{y}, \mathbf{x})$ is not decreased.

In both cases we proved that it is possible to modify strategy $\mathbf{y}$ such that either t decreases by 1 or $(h_i - \eta_i) + (h'_j - \eta'_j)$ decreases while t is not changed, which is a contradiction with the assumptions that we had on strategy $\mathbf{y'}$. Thus, strategy $\mathbf{y}$ the output of Algorithm 11 is indeed the best fractional response of the second player □

To give a 2-approximation of the first player's best 2-strategy, we first define the greedy opponent. The exact definition is given in Definition 99. His response is to find a fractional response using Algorithm 11 and round it down to an integral one. In Observation 100 we prove that the greedy opponent's response differs from the fractional response in at most two battlefields. Note that if $\mathsf{w}_{\mathsf{max}} > u/3$ holds, the first player simply achieves $(u/3, p)$-maximin 2-strategy by putting all his troops on the battlefield with wight $\mathsf{w}_{\mathsf{max}}$. For simplicity, throughout the chapter we assume $\mathsf{w}_{\mathsf{max}} \leq u/3$ holds. This yields that the best 2-strategy of the first player against such an opponent is a $(u/3, p)$-maximin 2-strategy. Thus, in the rest of this section we focus on finding the best strategy of player 1 against the greedy opponent.

Definition 99 (Greedy Opponent)**.** *Greedy opponent is a weaker version of the second player. The response of this opponent against a* 2-strategy *of the first player,* $\mathbf{s} = (\mathbf{x}, \mathbf{x'})$*, is as follows. Using Algorithm 11, he first finds a best fractional response of the second player against* $\mathbf{s}$ *, which is denoted by vectors* $\mathbf{h}$ *and* $\mathbf{h'}$*. Since it is a fractional response some of the elements in these vectors are fractional. The greedy opponent rounds down the elements of these vectors and plays according to the rounded vectors.*

Observation 100. *Let strategy* $\mathbf{s}$ *be the fractional response of player 2 against a given* 2-strategy *of the first player that is obtained from Algorithm 11. Also, let* $\mathbf{s'}$ *denote the response of the greedy opponent against the same strategy of the first player. Strategies* $\mathbf{s}$ *and* $\mathbf{s'}$ *put different amount of troops in at most two battlefields.*

Proof. We prove in Lemma 101 that there are at most two battlefields $i, j \in [k]$ where h_i, h_j, h'_i or h'_j are fractional. Since the greedy opponent rounds down these vectors, his strategy differs from the fractional strategy in at most two battlefields. □

In the following lemma we present three import properties of Algorithm 11 which we use later to prove the main theorem of this section.

Lemma 101. *Consider* a, b *and* c *that are defined in lines 4, 5 and 6 of Algorithm 11.*

1. *At the beginning of each iteration of Algorithm 11, any fractional element of* $\mathbf{h}$ *and* $\mathbf{h'}$ *is in* $\{h_a, h'_b, h_c, h'_c\}$*.*
2. *Also, at the beginning of each iteration at most one of* h_a, h'_b, h_c *or* h'_c *is fractional.*

3. *After the last iteration of the algorithm there are at most two indices $i, j \in [k]$ where h_i, h_j, h'_i or h_j is fractional.*

Proof. We first prove that at the beginning of the algorithm, all the elements of $\mathbf{h}$ and $\mathbf{h'}$ other than h_a, h'_b, h_c, and h'_c are integral. Let r, r' respectively denote the minimum ratios among the available elements of vectors $\mathbf{h}$ and $\mathbf{h'}$ in an arbitrary iteration of the algorithm. It is easy to see that neither r nor r' decreases throughout the algorithm. Thus, any element that is increased before this iteration is either unavailable or is in $\{h_a, h'_b, h_c, h'_c\}$. Also note that any element that has value less than 1 is either available or jointly available. Therefore, any element that is not in $\{h_a, h'_b, h_c, h'_c\}$ is equal to either 0 or 1.

Moreover, we show that at the beginning of each iteration of Algorithm 11, at most one of h_a, h'_b, h_c and h'_c is fractional. We use proof by induction. We prove that if it is not the last iteration of the algorithm, and at the beginning of that at most one element in $\{h_a, h'_b, h_c, h'_c\}$ is fractional, after this iteration the same holds. It suffices since we know that any fractional element is in $\{h_a, h'_b, h_c, h'_c\}$. Note that since at most one element in both vectors is fractional this fractional element is independently available. Therefore, it is either h_a or h'_b. If $r_a + r'_b \leq r_c + r_c$ then the algorithm increases h_a and h'_b until one of them is not available anymore (note that it is not the last iteration) which means it is equal to 1. Otherwise, the algorithm increases both h_c and h'_c by the same amount until one of them becomes equal to 1. Therefore, in both cases at most one element remains fractional.

Now, consider the last iteration of the algorithm. We proved that at the

beginning, at most one element in $\{h_a, h'_b, h_c, h'_c\}$ is fractional. Also, the fractional element is either h_a or h'_b. If at this iteration, the algorithm changes h_a and h'_b, then these are the only ones that can be fractional. Also, if the algorithm changes h_c and h'_c, then only h_c, h'_c and one of h_a or h'_b can be fractional. In both cases there are at most two battlefields i and j where h_i, h_j, h'_i or h'_j are fractional. □

For any 2-strategy of the first player we define a signature which is determined by Algorithm 11 and the state in which this algorithm terminates.

Definition 102 (Signature of a 2-strategy)**.** *For any strategy $s = (\mathbf{x}, \mathbf{x'})$ find the best fractional response of the algorithm using Algorithm 11 and denote it by vectors* $\mathbf{h}$ *and* $\mathbf{h'}$*. consider sets B_1, B'_1, and B_2 in the last iteration of the algorithm. Define*

- $a := \text{argmin}_{i \in B_1} r_i$,
- $b := \text{argmin}_{i \in B'_1} r'_i$, *and*
- $c := \text{argmin}_{i \in B_2} r_i + r'_i$.

Also, let μ be the total number of troops that are spent on h_a, h'_b, h_c and h'_c. For instance, if a, b and c are three different battlefields, $\mu := c_a \cdot h_a + c'_b \cdot h'_b + c_c \cdot h_c + c'_c \cdot h'_c$. Moreover, denote by u_1 and u_2 the utility that the greedy opponent achieves against strategies $\mathbf{x}$ *and* $\mathbf{x'}$ *in these battlefields. We define the signature of strategy s to be $(a, b, c, \mu, x_{a,b,c}, x'_{a,b,c}, u_1, u_2)$ and we denote it by σ_s. Note that we represent (x_a, x_b, x_c) by $x_{a,b,c}$.*

Lemma 103. *Let σ be the signature of the greedy opponent's response against a 2-strategy of the first player $(\mathbf{x}, \mathbf{x'})$. There exists a function that determines the response of the greedy opponent in any battlefield $i \in [k] - \{a, b, c\}$ given x_i, x'_i and*

σ.

Proof. Let σ be $(a, b, c, \mu, x_{a,b,c}, x'_{a,b,c}, u_1, u_2)$. Also without loss of generality, we assume that $x_i \leq x'_i$: therefore, $c_i = x_i$ and $c'_i = x'_i - x_i$ hold. We claim that Algorithm 12 finds h_i and h'_i given x_i, x'_i, and signature σ. To prove that this algorithm is correct, we need to show that if h_i or h_j is set to 1 by this algorithm, any greedy response to a strategy with this signature also sets this to 1. We also need to prove that any element that is set to 0 is also 0 any greedy response to a strategy with this signature.

Note that the minimum ratio of the available (or jointly available) elements does not decrease after each iteration. Also, note that in Algorithm 11, the tie breaking rule while finding the available element with the minimum ratio is the index of elements. Thus, the definition of the signature directly results that any element that is set to 1 by this algorithm is indeed correct. We just need to show the second part. By Algorithm 11, for battlefield i, if $r_i + r'_i > r_c + r'_c$ holds or $r_i + r'_i = r_c + r'_c$ and $i > c$ hold, $h'_i = 0$. Also, if in addition to the mentioned condition, $r_i > r_a$ holds and $r_i = r_a$ and $i > a$ hold then, $h_i = 0$. The only remaining case is when $r_i < r_a$ and $r'_i \geq r'_b$ holds. In this case the algorithm sets $h'_i = 0$ even if $r_i + r'_i \leq r_c + r'_c$. We prove by contradiction that $h'_i = 0$ is correct in this situation. Assume that there exists a response in which $h'_i = 1$. Let γ and γ' respectively denote the minimum ratios among the ratio of available elements in vectors $\mathbf{h}$ and $\mathbf{h'}$ at the iteration that h'_i first changes. The only way that h'_i changes is as a jointly available element. For this to be possible, conditions $r_i + r'_i \leq \gamma + \gamma'$

and $r_i + \gamma' > \gamma + \gamma'$ must hold. This is a contradiction with the fact that minimum ratio of the available elements does not decrease throughout the algorithm. Thus, Algorithm 12 correctly finds the best response of the greedy opponent in battlefield i. □

Algorithm 12: Greedy opponent's response in battlefield i given x_i, x'_i and $\sigma = (a, b, c, \mu, x_{a,b,c}, x'_{a,b,c}, \dot{u}_1, \dot{u}_2)$

Without loss of generality, this algorithm assumes that $x_i \leq x'_i$. Also r_i and r'_i are i-th elements of ratio vectors that can be computed using x_i, x'_i and w_i.

$h_i, h'_i \leftarrow 0$
if $r_i < r_a$ **or** $(r_i = r_a, i < a)$ **then**
    $h_i \leftarrow 1$
    if $r'_i < r'_b$ **or** $(r'_i = r'_b, i < b)$ **then**
        $h'_i \leftarrow 1$
    end if
else if $r_i + r'_i < r_c + r'_c$ **or** $(r_i + r'_i = r_c + r'_c, i < c)$ **then**
    $h_i, h'_i \leftarrow 1$
end if
return h_i and h'_i

Lemma 104. *Let S be the set of 2-strategies of the first player in an instance of Discrete Colonel Blotto and let σ_S denote the set of signatures of the strategies in S. Size of the set σ_S is polynomial and there exists a polynomial time algorithm to find it.*

Proof. Let $(a, b, c, \mu, x_{a,b,c}, x'_{a,b,c}, \dot{u}_1, \dot{u}_2)$ denote the signature of an arbitrary 2-strategy $s = (\mathbf{x}, \mathbf{x'})$. Note that all possible values of $a, b, c, x_{a,b,c}$ and $x'_{a,b,c}$ is bounded by a polynomial in n and k. Also, by Lemma 101, the amounts of troop that the second player spends in the rest of the battlefields is integral. Therefore, μ is polynomial in m. Thus, we just need to prove that given an assignment to these variables there are polynomially many possible cases for values of $\dot{u}_1$ and $\dot{u}_2$. Let u_1 and u_2 denote the

amount of troops that the the greedy opponent gets in the other $k-3$ battlefields. By Lemma 97, in a valid response of this opponent, $u_1 + \dot{u}_1 = \dot{u}_2 + u_2$ holds. Since $u_1 - u_2$ has polynomially many possible values, the same holds for $\dot{u}_1 - \dot{u}_2$. We claim that there is a polynomial time algorithm that finds $\dot{u}_1$ and $\dot{u}_2$ given value of $\dot{u}_1 - \dot{u}_2$. By Lemma 101, before the last iteration of Algorithm 11, at most one of h_a, h'_b, h_c and h'_c is fractional and all the other elements are integral. We also mention in the proof that this fractional one is either h_a or h'_b. This element is responsible for the difference between $\dot{u}_1$ and $\dot{u}_2$; therefore, the value of that is uniquely determined given $\dot{u}_1 - \dot{u}_2$. After fixing the value of this fractional element, we find the exact values of $\dot{u}_1$ and $\dot{u}_2$ by simulating the last iteration of the algorithm. □

Theorem 105. *Given that there exists a (u, p)-maximin 2-strategy of the first player, there is an algorithm that finds a $(u/3, p)$-maximin 2-strategy of this player in polynomial time.*

Proof. By Observation 100, the strategy of the greedy opponent differs from the fractional best response in at most two battlefields. Also, the best integral strategy gets utility at most equal to the fractional one. Therefore, this weaker opponent loses at most $2 \cdot \mathsf{w}_{\mathsf{max}}$ compared to his best response strategy. Note that if losing at most $2 \cdot \mathsf{w}_{\mathsf{max}}$ does not guarantee a $(u/3, p)$-maximin 2-strategy for player 1, he can put all his troops in the battlefield with the maximum weight and just win that one. This yields that the best strategy of the first player against the greedy opponent guarantees a $(u/3, p)$-maximin 2-strategy for him.

The problem that we need to solve now is how to find the best strategy of

player 1 against the greedy opponent. We cannot afford to go over all the possible strategies and search for the best one. However, playing against the greedy opponent gives us the ability to narrow down our search space and find the best response by solving polynomially many dynamic programs. For any signature σ, let S_σ denote the set of all the first player's strategies with signature σ. To find the best strategy of player 1 against the greedy opponent, we just need to exhaustively searches through all the possible signatures and for any σ find the best strategy of player 1 in set S_σ. Note that by Lemma 104 there are polynomially many valid signatures. We give a dynamic program that given a signature $\sigma = (a, b, c, \mu, x_{a,b,c}, x'_{a,b,c}, \dot{u_1}, \dot{u_2})$, finds the best strategy in set S_σ. By Lemma 103, there exists a function that determines the response of the greedy opponent in any battlefield i given x_i, x'_i, and σ. Let $f(i, x_i, x'_i, \sigma)$ denote this function. The dynamic program that we use is $D(j, n_1, n_2, \omega, u_1, u_2)$ with $j \in \{0, \ldots, k-3\}$, $n_1, n_2 \in \{0, , \ldots, n\}$, $\omega \in \{0, 1, \ldots, m-\mu\}$ and $u_1, u_2 \in \{0, \ldots, \sum_i w_i\}$. Note that j is in set $\{0, \ldots, k-3\}$ since before the dynamic program we remove three battlefields a, b and c. The value of $D(j, n_1, n_2, \omega, u_1, u_2)$ is either 0 or 1. It is 1 iff it is possible to give two partial strategies $\mathbf{x} = (x_1, \ldots, x_j)$ and $\mathbf{x'} = (x'_1, \ldots, x'_j)$ over the first j battlefields such that all the following conditions are satisfied. Let $\mathbf{h}$ and $\mathbf{h'}$ denote the partial response of the greedy opponent to 2-strategy $(\mathbf{x}, \mathbf{x'})$ where for any i, h_i and h'_i are determined by $f(i, x_i, x'_i, \sigma)$.

- Condition 1: $\Sigma_{i\in[j]} x_i = n_1$ and $\Sigma_{i\in[j]} x'_i = n_2$. This is a condition on the amount of troops that the first player uses in the first j battlefields.

- Condition 2: $\mathbf{w} \cdot \mathbf{h} = u_1$ and $\mathbf{w} \cdot \mathbf{h'} = u_2$. This condition guarantees that the utility that the greedy opponent gets against strategies $\mathbf{x}$ and $\mathbf{x'}$ in the first j battlefield is respectively u_1 and u_2.

- Condition 3: $\mathbf{c} \cdot \mathbf{h} + \mathbf{c'} \cdot \mathbf{h'} = \omega$, where $\mathbf{c'}$ and $\mathbf{c}$ are partial cost vectors of $\mathbf{x}$ and $\mathbf{x'}$. This condition guarantees that the greedy opponent uses ω troops in the first j battlefields.

Base case. We start with the case where $j = 0$. Here, clearly, $D(0, n_1, n_2, \omega, u_1, u_2) = 1$ iff n_1, n_2, ω, u_1 and u_2 are all 0.

Updating the DP. To update $D(j, n_1, n_2, \omega, u_1, u_2)$, we only have to decide on how many troops to put on the j-th battlefield in strategies $\mathbf{x}$ and $\mathbf{x}'$. We try all the possibilities for values of variables x_j and x'_j and check, recursively, whether any of these choices satisfies the requirements for the dynamic value to be 1. Let ξ and ξ' denote an assignment to x_j and x'_j. Also, let ω' denote the amount of troops that the greedy opponent puts in battlefield j if $x_j = \xi$ and $x'_j = \xi'$. Moreover, u'_1 and u'_2 respectively denote the utility that he gets against strategies $\mathbf{x}$ and $\mathbf{x}'$ in this battlefield. Note that u'_1, u'_2 and ω' can be achieved using function f. $D(j, n_1, n_2, \omega, u_1, u_2) = 1$ iff there exist at least a pair of $\xi \in \{0, \ldots, n_1\}$ and $\xi' \in \{0, \ldots, n_2\}$ where

$$D(j-1, n_1 - \xi, n_2 - \xi', \omega - \omega', u_1 - u'_1, u_2 - u'_2) = 1.$$

After finding the value of D for all the valid inputs, we need to identify the strategies that are in set S_σ to be able to find the best one. Roughly speaking, one

may think that to find the best strategy of player 1 in set S_σ, it is enough if using the dynamic data we find the strategy that minimizes the utility that the greedy opponent gets in $k-3$ remaining battlefields. However, it is not true since using this method we may end up finding a strategy that is apparently very good for the first player but does not have the same signature as σ. This may happen since in the dynamic we do not consider the fact that the greedy opponent finds a strategy in which he gets the same utility against both strategies $\mathbf{x}$ and $\mathbf{x'}$. To avoid that, before finding the best strategy using the dynamic data we need to filter out the strategies that are not in S_σ. Recall that signature σ is $(a, b, c, \mu, x_{a,b,c}, x'_{a,b,c}, \dot{u_1}, \dot{u_2})$. For any pair of u_1 and u_2 that $D(k-3, n_1, n_2, m-\mu, u_1, u_2) = 1$ there exists a 2-strategy $\mathbf{s} \in S_\sigma$ against which the greedy opponent gets utility $u_1 + \dot{u_1}$ iff $u_1 + \dot{u_1} = u_2 + \dot{u_2}$ since any response of the greedy opponent to a 2-strategy $\mathbf{s} = (\mathbf{x}, \mathbf{x'})$ achieves the same utility against both strategies $\mathbf{x}$ and $\mathbf{x'}$. Therefore, to find the best 2-strategy in S_σ, using the dynamic data we find a pair of u_1 and u_2 such that $D(j, n_1, n_2, \omega, u_1, u_2) = 1$ and $u_1 + \dot{u_1} = u_2 + \dot{u_2}$ subject to minimizing $u_1 + \dot{u_1}$. This means that there exists a 2-strategy $\mathbf{s}$ of the first player against which the greedy opponent achieves $u_1 + \dot{u_1}$ utility.

By Lemma 104, there are polynomially many different valid signatures and there is a polynomial time algorithm to find them. Therefore, one can find the best strategy of the first player against the greedy opponent by going over all the possible signatures and finding the best 2-strategy using the described dynamic program. Note that having a best strategy against the greedy opponent gives a $(u/3, p)$-maximin 2-strategy of player 1. □

5.4.2.2 A $(1-\epsilon)$-Approximation

In Section 5.4.1 we give a $(1-\epsilon)$-Approximation for the case of one pure strategy. Here, we use the same idea and adapt it for our purpose. Recall that, we first define an updated wight vector by rounding down the weight of battlefields to a power of $(1+\epsilon)$. Then, we partition them to two sets of heavy and light battlefields with threshold $\tau = \epsilon u/4$. Corollary 86 implies that finding a $((1-\epsilon/2)u, 1)$-maximin 2-strategy after this modification gives us a $((1-\epsilon)u, 1)$-maximin 2-strategy. Therefore, we only focus on the instance with the updated weights. The overall idea is to first define a weaker opponent such that finding the best strategy of the first player against him gives us a $((1-\epsilon/2)u, 1)$-maximin 2-strategy. Then, give a polynomial time algorithm that finds this best strategy. The weaker adversary in the previous section is an opponent whose response against a strategy of the first player is to go over all his pure strategies on the heavy battlefields and for each one, play greedily on the light battlefields. The weaker adversary that we define against 2-strategies is almost similar to the one in the previous section. The only difference is on the greedy algorithm that he uses on the light battlefields which is to respond as a greedy opponent (with a minor technical modification that we explain later). Note that the greedy opponent loses at most $2 \cdot \mathsf{w}_{\mathsf{max}}$ compared to his best response; thus, playing against this weaker opponent gives a $((1-\epsilon/2)u, 1)$-maximin 2-strategy of the first player.

To be able to provide a best response algorithm, we first give a new representations for the strategies of both players on heavy battlefields so that we can

give proper limits on the number of strategies that they have. By Observation 87, the number of distinct heavy battlefields in bounded by a constant. For the case of 1-strategies, we represent first player's strategies by the number of troops that he puts on each battlefield weight. However, the exact same representation does not work for 2-strategies of the first player, but we are still able to represent them by vectors of polynomial length such that the number of valid representations is polynomial as well. Note that given a 2-strategy $s = (\mathbf{x}, \mathbf{x'})$ of the first player, for each battlefield $i \in [k]$, either $x_i \leq x'_i$ or $x_i > x'_i$ holds. Based on this we partition the battlefields to two types. A battlefield i is of type 1 if $x_i \leq x'_i$ otherwise it is of type 2. Lemma 106 proves that there is an optimal 2-strategy $s = (\mathbf{x}, \mathbf{x'})$ of the first player where both strategies $\mathbf{x}$ and $\mathbf{x'}$ put roughly the same number of troops in battlefields of the same type and the same weight.

Lemma 106. *If player 1 has a (u, p)-maximin 2-strategy , he also has a (u, p)-maximin 2-strategy $s = (\mathbf{x}, \mathbf{x'})$ where for any two battlefields i and j that conditions $w_i = w_j$, $i \leq j$, and $x_i \leq x'_i$ hold we have $0 \leq x_i - x_j \leq 1$ and $0 \leq x'_i - x'_j \leq 1$.*

Proof. Let $s = (\mathbf{x}, \mathbf{x'})$ be a (u, P)-maximin 2-strategy of player 1. Assume battlefields i and j, where $i < j$, are of the same type and the same weight, but conditions $|x_i - x_j| \leq 1$ and $|x'_i - x'_j| \leq 1$ do not hold for them. Let $t = x_i + x_j$ and $t' = x'_i + x'_j$ respectively denote the amount of troops that strategies $\mathbf{x'}$ and $\mathbf{x'}$ put on these battlefields. There are five possible cases for the utility that the second player gets in these battlefields. We show that it is possible to redistribute t and t' on battlefields i and j to satisfy the mentioned conditions without lowering the number of troops

that the opponents needs to spend in each case. Let u and u' respectively denote the utility that the second player gets against strategies $\mathbf{x}$ and $\mathbf{x'}$ in these battlefields. All the possible cases for value of u and u' are as follows.

1. Case 1: $u = w$ and $u' = w$. The second player achives this by spending $\min(x'_i, x'_j)$ troops which is maximized when $x'_i = \lceil t'/2 \rceil$, $x'_j = \lfloor t'/2 \rfloor$.

2. Case 2: $u = 2w$ and $u' = w$. The opponent spends $\min(x'_i + x_j, x'_j + x_i)$ troops to get this utility. This is maximized when $x'_i = \lceil t'/2 \rceil$, $x'_j = \lfloor t'/2 \rfloor$, $x_i = \lceil t/2 \rceil$ and $x_j = \lfloor t/2 \rfloor$.

3. Case 3: ($u = 0$ and $u' = 0$), ($u = w$ and $u' = w$) or ($u = 2w$ and $u' = 2w$). These cases are independent of the distribution. The opponent spends respectively 0, t and $t + t'$ troops to achieve theses results.

Therefore, if a given strategy does not satisfy the desired conditions, we modify it by setting $x'_i = \lceil t'/2 \rceil$, $x'_j = \lfloor t'/2 \rfloor$, $x_i = \lceil t/2 \rceil$ and $x_j = \lfloor t/2 \rfloor$. As we showed, after this modification there is no possible amount of utility that the second player can get with spending fewer number of troops in battlefields i and j. Also this modifidied strategy satisfies $0 \leq x_i - x_j \leq 1$ and $0 \leq x'_i - x'_j \leq 1$. □

We represent any partial 2-strategy of the first player on the heavy battlefields by a vector of length $\mathsf{poly}(1/\epsilon)$. Each entry of this vector is associated with a heavy weight. The i-th entry consists of a tuple which determines how many battlefields of the i-th heavy weight are from type 1, and how many troops each of $\mathbf{x}$ and $\mathbf{x'}$ spend on the battlefields of each type with this weight. Let us denote by $\mathcal{S}_1^h$ the

set of all such vectors. Note that each entry of a vector in this representation has polynomially many possible values. The following is a corollary of this fact.

Corollary 107. $|\mathcal{S}_1^h| \leq n^{\mathsf{poly}(1/\epsilon)}$ *where* $\mathcal{S}_1^h$ *is the set of* 2-strategies *of the first player on the heavy battlefields.*

Also, using the same argument as we had for the case of 1-strategies, one can see that, here as well, it is possible to represent pure strategies of the second player on heavy battlefields by vectors of constant size. Let $\mathcal{S}_2^h$ denote the set of all such vectors. It is easy to see that bservation 87 holds for this case as well, and $|\mathcal{S}_2^h|$ is bounded by a constant.

As we mentioned, the weaker adversary acts as a greedy opponent on the light battlefields; therefore, he uses Algorithm 11 to find a best fractional response on these battlefields. However, there is a technical detail that we should consider here. Let $s = (\mathbf{x}, \mathbf{x'})$ denote the strategy of player 1 and let $\mathbf{y}$ be a partial response of the weaker adversary on heavy battlefields. By Lemma 97 any best fractional response gets the same utility against both the strategies of the first player. Note that it is possible that $\mathsf{u}_2(\mathbf{x}, \mathbf{y}) = \mathsf{u}_2(\mathbf{x'}, \mathbf{y})$ does not hold. Thus, the amount of utility that a best fractional response gets against $\mathbf{x}$ and $\mathbf{x'}$ on the light battlefields differs by $\mathsf{u}_2(\mathbf{x}, \mathbf{y}) - \mathsf{u}_2(\mathbf{x'}, \mathbf{y})$. Here, the issue is that in Algorithm 11 we assume that at the beginning the amounts of utility that the greedy opponent has against strategies $\mathbf{x}$ and $\mathbf{x'}$ are both equal to 0. Therefore, at each iteration the algorithm increases them by the same amount. Hence, the same algorithm cannot be used here, but with a slight modification we can use it for the case of $\mathsf{u}_2(\mathbf{x}, \mathbf{y}) \neq \mathsf{u}_2(\mathbf{x'}, \mathbf{y})$ as well. Denote

by u and u' the amount of utility that the fractional response gets against strategies $\mathbf{x}$ and $\mathbf{x'}$ until an arbitrary iteration of Algorithm 11. If we use it to complete the partial strategy $\mathbf{y}$ on the light battlefields, at the beginning $u = \mathsf{u}_2(\mathbf{x}, \mathbf{y})$ and $u' = \mathsf{u}_2(\mathbf{x'}, \mathbf{y})$. Without loss of generality, assume $u' \leq u$. The modification is as follows. Since we want u and u' to be equal at the end, before we start increasing both of them, we greedily increase the available elements of vector $\mathbf{h}$ (to increases u) until $u = u'$ holds. This is a correct approach since in a best fractional response the overall weight of the light battlefields in which $h_i > h'_i$ is at least $u - u'$. Also, any such battlefield is available at the beginning of Algorithm 11. One can easily verify that this modification does not affect the properties that we have proved for the greedy opponent's response.

Also, recall that signature of any 2-strategies of the first player, defined in Definition 102, is unique since the response of the greedy opponent against a 2-strategy is unique as well. However, the signature of a 2-strategy on the light battlefields also depends on the strategy of the opponent on the heavy battlefields. Roughly speaking, a 2-strategy of the first player $s = (\mathbf{x}, \mathbf{x'})$, for any $\mathbf{y^h} \in \mathcal{S}_2^h$ faces a different type of greedy opponent on the light battlefields since the remaining amount of troops and $\mathsf{u}_2(\mathbf{x}, \mathbf{y^h}) - \mathsf{u}_2(\mathbf{x'}, \mathbf{y^h})$ changes the strategy of the greedy opponent on the light battlefields. Therefore, in the definition of signature of a 2-strategy on the light battlefields we should also include the type of the opponent which is determined by a strategy of the second player on the heavy battlefields. Given any such strategy, $\mathbf{y^h}$, we denote signature of 2-strategy $\mathbf{s}$ by $\sigma(\mathbf{s}, \mathbf{y^h})$. Now, we are ready to prove the main theorem of this section which is as follows.

Theorem 96 (restated). *Given that player 1 has a* (u, p)-*maximin* 2-*strategy, there exists a polynomial time algorithm that obtains a* $((1-\epsilon)u, p)$-*maximin* 2-*strategy of player 1 for any arbitrarily small constant* $\epsilon > 0$.

Proof. Let $s^h = (\mathbf{x}, \mathbf{x'}) \in \mathcal{S}_1^h$ be a 2-strategy of the first player on heavy battlefields and let $S(s^h)$ denote the set of all the 2-strategies of the first player that plays strategy $\mathbf{s^h}$ on heavy battlefields. We find a partial 2-strategy on the light battlefields $\mathbf{s^l}$ such that the combination of these 2-strategies, denoted by $\mathbf{s} = (\mathbf{s^h}, \mathbf{s^l})$, loses at most $\epsilon u/2$ compared to the best strategy in $S(s^h)$. For any $i \in |\mathcal{S}_2^h|$, strategy $\mathbf{y_i^h}$ denotes the i-th strategy in $\mathcal{S}_2^h$. For any $i \in |\mathcal{S}_2^h|$, we fix a signature σ_i and find a 2-strategy $\mathbf{s^l}$ such that for any i, $\sigma((\mathbf{s^h}, \mathbf{s^l}), \mathbf{y_i^h}) = \sigma_i$ is satisfied. Also, among all such strategies, we pick the one that minimizes the utility of the weaker adversary against $(\mathbf{s^h}, \mathbf{s^l})$. Recall that to find a best strategy of the first player against a single greedy opponent we design a dynamic program in the proof of Theorem 105. The main difference here is that we want a strategy that is the best against multiple greedy opponents; therefore, we need to run these dynamic programs in parallel and update them simultaneously. For that, we use an idea that is presented in the previous section. In Section 5.4.1, we design a dynamic program that finds the best strategy of the first player against multiple weaker adversaries at the same time. Combining this idea with the dynamic program presented in the proof of Theorem 105 gives us a polynomial time algorithm to find a best strategy of the first player against the weaker adversary.

Note that $|\mathcal{S}_2^h|$ is bounded by a constant and by Lemma 104 the number

of possible signatures is polynomial; thus, all the possible cases that we consider to find the best strategy of the first player against the weaker adversary in $S(s^h)$ is polynomial as well. Also, size of the set $\mathcal{S}_1^h$ is polynomial. Hence, we can find the best 2-strategy of the first player against the weaker adversary in polynomial time. As we mentioned such a strategy is a $((1-\epsilon)u, p)$-maximin 2-strategy strategy of the first player. □

5.4.3 Generalization to the Case of c-Strategies for $c > 2$

In this section the goal is to provide an algorithm that finds a $((1-\epsilon)u, p)$-maximin c-strategy of the first player given that the existence of a (u, p)-maximin c-strategy is guaranteed. The result of this section is stated in the following theorem more formally.

Theorem 108. *Given that there exists a (u, p)-maximin c-strategy of player 1, there is a polynomial time algorithm that finds a $((1-\epsilon)u, p)$-maximin c-strategy of this player.*

Note that in the case of (u, P)-maximin 2-strategy we show that in an optimal strategy the first player plays both strategies with the same probability. It is easy to see that for the case of $c > 2$ this does not hold. However, we show that to find an optimal strategy it suffice to just consider a constant set of probability assignment to the strategies. By Theorem 77 given that a (u, p)-maximin c-strategy is guaranteed, there exists an algorithm to construct a set P_c of $O(1)$ profiles in time $O(1)$. Recall that for a mixed strategy $\mathbf{x}$, its profile, denoted by $\rho(\mathbf{x})$, is a multisite of probabilities

associated to the pure strategies in the support of $\mathbf{x}$. Since $|P_c|$ is bounded by a constant we can solve the problem for all profiles P_c and for each one find the best strategy of player 1 that has this profile. In the rest of this section we assume that a probability assignment to the strategies is given. Let $\mathbf{s} = (\mathbf{x^1}, \dots \mathbf{x^c})$ be a c-strategy of the first player. For any $i \in [c]$ denote by p^i the probability with which the first player plays strategy $\mathbf{x^i}$.

Similar to the case of $((1-\epsilon)u, p)$-maximin 2-strategy, we start by decomposing the battlefields to sets of heavy and light battlefields. the threshold for that is $\tau = \epsilon u/2c$. We also round down the wights as mentioned in the previous sections and just solve the problem for rounded wights. Then, we give proper limits on the number of possible c-strategies of the first player and the strategies of the second player on heavy battlefields to be able to design a polynomial time dynamic program that finds this a $(1-\epsilon)$-approximate c-strategy. Let $\mathbf{s} = (\mathbf{x^1}, \dots, \mathbf{x^c})$ be a c-strategy of the first player. We say two heavy battlefields i and j are from the same type if $w_i = w_j$ and there exists a permutation $(a_1, a_2, \dots, a_c)$ of numbers 1 to c where $x_i^{a_1} \leq \dots \leq x_i^{a_c}$ and $x_j^{a_1} \leq \dots \leq x_j^{a_c}$ hold. We claim if a (u, p)-maximin c-strategy of the first player exists, there also exists a (u, p)-maximin c-strategy in which the number of troops that the first player puts in the battlefields of the same type in each strategy is almost the same. The formal claim is stated in the following lemma.

Lemma 109. *If player 1 has a (u, p)-maximin c-strategy, he also has a (u, p)-maximin c-strategy $(\mathbf{x^1}, \dots \mathbf{x^c})$ where for any two battlefields of the same type i and j and any $a \in [c]$ we have $0 \leq x_i^a - x_j^a \leq 1$.*

Proof. Assume a best (u, p)-maximin c-strategy $\mathbf{s}$ is given that does not satisfy condition $0 \leq x_i^a - x_j^a \leq 1$ for some $a \in [c]$. For any strategy $\mathbf{x^a}$, let t^a denote the number of troops that this strategy spends in battlefields i and j. We modify $\mathbf{s}$ by redistributing these troops. For any $a \in [c]$, set $x_j^a = \lfloor t^a/2 \rfloor$ and $x_i^a = \lceil t^a/2 \rceil$. We prove that this modification does not lower the amount of troops that the opponent needs to spend in these two battlefields to get any possible amounts of utility from strategies $\mathbf{x^1} \dots \mathbf{x^c}$. Let w denote the weight of battlefields i and j. Also we say a strategy x^a is smaller than x^b in these battlefields if a comes before b in the given permutation (that defines their type). Utility of the second player in these battlefields has the following form: there are two strategies $\mathbf{x^a}$ and $\mathbf{x^b}$ where $\mathbf{x^a} < \mathbf{x^b}$ and the second player gets utilities $2 \cdot w$ against strategies smaller than or equal to $\mathbf{x^a}$ and gets utility w against the ones that are between $\mathbf{x^a}$ and $\mathbf{x^b}$. To get this utility the second player spends $\min(x_i^a + x_j^b, x_j^a + x_i^b)$ number of troops. One can easily verify that our modification does not decrease this. Therefore, any given (u, p)-maximin c-strategy can be transformed to a c-strategy that gets the same utility and satisfies the mentioned conditions. □

Note that there are $\mathsf{poly}(1/\epsilon)$ different weights of heavy battlefields. Also, the number of permutation of numbers 1 to c is $c!$. Therefore, there are $\mathsf{poly}(1/\epsilon)$ types of heavy battlefields. This means that we can represent any c-strategy of the first player on the heavy battlefields by a vector of length $\mathsf{poly}(1/\epsilon)$. Each element of this vector, for any $i \in [c]$, contains a variable that shows how many troops strategy $\mathbf{x^i}$ puts in battlefields of this type. Let $\mathcal{S}_1^h$ denote the set of all such partial c-strategies

in this representation. Since the total number of troops is n, each variable has n possible values. The following is a corollary of this.

Corollary 110. $|\mathcal{S}_1^h| \leq n^{\mathsf{poly}(1/\epsilon)}$.

When the goal of the first player is to achieve utility u with probability p, he loses if the second player gets utility at least u with probability more than $1-p$. We call a set of strategies in $x^1, \ldots x^c$ a *losing set* if the sum of probability of the strategies in it is more than $(1-p)$. If the second player gets utility at least u against all the strategies in at least one losing set, the first player loses in the sense that he can not get utility u with probability p. Therefore, a c-strategy of the first player is a (u,p)-maximin c-strategy iff there does not exist a pair of a losing set L and a pure strategy of the second player $\mathbf{y}$ where $\mathbf{y}$ gets utility at least u against all the strategies in L. Let L_c be the set of all the losing sets. For any $L \in L_c$ we define a new type of opponent whose goal is to get utility at least u against all the pure strategies in set L and we denote it by P_L. Also, recall that to prevent player 1 from achieving a payoff of u, any opponent of type P_L can only lose in at most $2c/\epsilon$ heavy battlefields against any strategy in L. We represent any pure strategy of this opponent by a vector of size $\mathsf{poly}(1/\epsilon)$. Entries of this vector represent the number of battlefields of any type that the second player wins against any strategy of the first player in set L. Therefore, each entry contains $c \cdot c!$ numbers where each one is in $[0, 2c/\epsilon]$. Let $|\mathcal{S}_2^h(L)|$ be the set of all partial pure strategies of the opponent of type P_L on heavy battlefields. We denote any pure strategy of the second player on heavy battlefields by a pair of $(L, \mathbf{y})$ where L determines the type of the opponent

and $\mathbf{y}$ is his partial strategy on the heavy battlefields. Let $|\mathcal{S}_2^h|$ denotes the set of all such pairs.

Corollary 111. $|\mathcal{S}_2^h| \leq O(1)$.

Similar to the Section 5.4.1 and Section 5.4.2 we define a weaker adversary such that his best strategy loses at most $u/2c$ compared to the best strategy of the second player. We also design a dynamic program that given a partial c-strategy of player 1 and a partial strategy of the weaker adversary on the heavy battlefields, finds the best c-strategy of player 1 against this opponent on the light battlefields. Without loss of generality we assume that the utility of the opponent is the minimum utility that he gets against all the strategies of the first player. The reason is that we have different types of opponents and for each one we fix the strategies that he is playing against. One can verify that combining these with the methods of the pervious sections gives us an algorithm to find a $((1-\epsilon)u, p)$-maximin c-strategy of the first player.

Recall that in the case of 2-strategies, we had an alternative representation of the second player's response which is a pair of binary vectors of length k. We define a similar representation for the response of the second player against c-strategies. Note that the main representation of his response is by a vector of length $[k]$ in which the i-th entry is the number of troops that the second player puts in the i-th battlefield. An alternative representation is to represent any response of player 2 against a c-strategy $\mathbf{s} = (\mathbf{x^1}, \ldots, \mathbf{x^c})$ of player 1 by c vectors of length k which we denote by $(\mathbf{h^1}, \ldots, \mathbf{h^c})$. For any $i \in [c]$, vector $\mathbf{h^i}$ determines the strategy of player

2 against strategy $\mathbf{x^i}$. If $h_b^i = 1$ holds for a battlefield $b \in [k]$, the second player wins strategy $\mathbf{x^i}$ in this battlefield. Note that if for a pair of $i, j \in [c]$, $x_b^i < x_b^j$ holds, then $h_b^j = 1$ yields $h_b^i = 1$ which means $h_b^j \leq h_b^i$. We also define c cost vectors $(\mathbf{c^1}, \ldots, \mathbf{c^c})$ for the strategy of first player. These cost vectors, are to transform strategies of player 2 between the two representations. For any battlefield b, let π^b be a permutation of numbers 1 to c where for any two consecutive elements i, j in that, $x_b^i \leq x_b^j$. We set $c_b^j := x_b^j - x_b^i$. Roughly speaking, assuming that the second player wins strategy $\mathbf{x^i}$ in battlefield b, entry c_b^j is the number of troops that he needs to add to this battlefield to win strategy $\mathbf{x^i}$ as well. Also, for $l := \pi_1^b$ we set $c_b^l := x_b^l$. A set of vectors is a fractional solution to the best response problem iff:

1. For any pair of $i, j \in [c]$, $x_b^i < x_b^j$ yields $h_b^j \leq h_b^i$.

2. Entries of the vectors are fractional numbers between 0 and 1.

3. The amount of troops used by the second player is at most m. In the other words, $\Sigma_{i \in [c]} \mathbf{c^i} \cdot \mathbf{h^i} \leq m$.

Such a response is a best response iff $\min_{i \in [c]} \mathbf{w} \cdot \mathbf{h^i}$ is maximized. We define the strategy of the weaker adversary as follows: he searches through all his pure strategies on the heavy battlefields and for each one, finds his best fractional response on the light battlefields. Then, he rounds down the fractional vectors and plays according to the rounded vectors on the light battlefields. Note that any integral response gets utility at most equal to the best fractional strategy. We prove that there exists a best fractional response that the number of battlefields in which at least one of

the vectors $\mathbf{h^1}, \ldots, \mathbf{h^c}$ is fractional is at most c. Thus, the weaker adversary loses at most $\epsilon u/2$ compared to the best integral strategy of the second player.

Lemma 112. *For any response of the second player* $\mathbf{y}$*, let* $B_{\mathbf{y}}$ *denote the set of battlefields where for any* $i \in B_{\mathbf{y}}$ *there exists at least one* $j \in [c]$ *where* h_i^j *is fractional. There exists a best fractional response of player 2 for which* $|B_{\mathbf{y}}| \leq c$*.*

Proof. Let $\mathbf{y}$ be a best fractional strategy with minimum $|B_{\mathbf{y}}|$. For any $i \in B_{\mathbf{y}}$ define a vector $\mathbf{v^i}$. For any $j \in [c]$, entry $v_j^i := 1$ if h_j^i is fractional. Otherwise it is 0. By the optimality of the solution, all such vectors are independent. Thus, the number of such vectors is bounded by the dimension which is c here. □

Let $\mathbf{s} = (\mathbf{x^1}, \ldots, \mathbf{x^c})$ be a strategy of player 1 and let $\mathbf{y}$ be the response of the weaker adversary to that. We define the signature of $\mathbf{s}$ to be the set of c battlefields that have at least a fractional element in $\mathbf{y}$ and the number of troops that each player puts in them. We claim that knowing the signature of a strategy and the number of troops that different strategies of player 1 put in a given battlefield, one can uniquely determine the strategy of player 2 in that battlefield. Define the ratio of a subset of strategies in $\mathbf{s}$ to be the minimum amount of troops that one need to add to the fractional solution to increase the utility of the second player against these strategies by a very small fixed amount denoted by δ. The overall idea is that given the signature of a strategy, we can find these ratios for all the subsets and similar to what we do in Algorithm 12 for the case of 2-strategies, these ratios and the index of the fractional battlefields are enough to determine the strategy of player 2 in a given battlefield. Having this function and combining it with the

ideas of the previous section (the dynamic program designed in the proof of 105) gives us a dynamic program to find the best strategy of player 1 against the weaker adversary.

5.5 Extension to Maximin Strategies

In this section, we show that our results carry over to the case where our goal is to maximize the guaranteed expected utility. Recall that for the case of (u, p)-maximin strategies, we proved in Section 5.2 that regardless of the game structure, it suffices to only consider a constant number of probability assignments (profiles) to the pure strategies. We used this to first fix the profile and then solve the game by finding the actual pure strategies. Unfortunately, this is not the case when the objective is to maximize the expected utility. However, we show that it is possible to consider only a polynomial number of profiles while ensuring that the found solution among them is a $(1-\epsilon)$-approximation of the actual maximin strategy.

Throughout this section we denote by OPT the guaranteed expected payoff of the optimal maximin c-strategy. Our goal is to construct a c-strategy in polynomial time that guarantees an expected utility of at least $(1-\epsilon)\mathsf{OPT}$ against any strategy of the opponent for any given constant $\epsilon > 0$. We call this a $(1-\epsilon)$-approximate maximin strategy. We start with the following claim.

Claim 113. *Either* $\mathsf{OPT} = 0$ *or* $\mathsf{OPT} > 1/c$.

Proof. Assume that $\mathsf{OPT} > 0$. This means that there exists a set S of c pure strategies, where against any strategy of player 2, at least one of the strategies in

S obtains a non-zero utility. Since the battlefield weights are integers, against any strategy of player 2, at least one strategy in S obtains a payoff of at least 1. Now, by playing each of these strategies with probability $1/c$, we guarantee an expected utility of at least $1/c$ against any strategy of the opponent. Hence $\mathsf{OPT} > 1/c$. □

Let us denote by $w := \sum_{i\in[k]} w_i$ the sum of all battlefield weights. Our next claim gives a lower bound for the probabilities assigned to the strategies in the support.

Claim 114. *For any given $\epsilon > 0$, there exists a $(1-\epsilon)$-approximate maximin strategy for any instance of (continuous or discrete) Colonel Blotto where every strategy in the support is played with probability at least $\frac{\epsilon\mathsf{OPT}}{cw}$.*

Proof. Consider an optimal maximin strategy. If no strategy in its support is played with probability less than $\frac{\epsilon\mathsf{OPT}}{cw}$, we are done. Otherwise, set the probability of all such strategies in the support to be 0 (i.e., remove them from the support). Now consider a strategy of player 2. Each of the removed strategies gets a utility of at most w against this strategy since w is sum of battlefield weights. On the other hand, there are at most c such strategies. Therefore, the overall cost for the expected utility is

$$\frac{\epsilon\mathsf{OPT}}{cw} \cdot c \cdot w = \epsilon\mathsf{OPT}. \tag{5.18}$$

This implies that the remaining strategies in the support obtain an expected utility of at least $(1-\epsilon)\mathsf{OPT}$ concluding the proof. □

Assuming that $\mathsf{OPT} > 0$ (otherwise a single pure strategy without any troops

is the solution), by combining the two claims above we get the following observation.

Observation 115. *We can assume w.l.o.g., that the minimum probability is* $\Omega(1/w)$.

We use this observation to consider only $O(\log w)$ probabilities for each strategy, leading to a polynomial number of profiles that have to be considered.

Lemma 116. *For any constant* $\epsilon > 0$*, and for any instance of continuous or discrete Colonel Blotto, there are only polynomially many profiles among which an* $(1-\epsilon)$*-approximate maximin is guaranteed to exist.*

Proof. Suppose that the probabilities are all in set $P = \{p_0, (1+\epsilon)p_0, (1+\epsilon)^2 p_0, \ldots, 1\}$ where p_0 is the lower bound for minimum probability. We showed in Observation 115 that it suffices to have $p_0 = \Omega(1/w)$, therefore $|P| \leq O(\log w)$ since ϵ is assumed to be constant. Note that $O(\log w)$ is polynomial in the input size, thus, even if we try $O(\log w)$ possibilities for c strategies, we have to try polynomially many possibilities. It remains to prove that P provides a $(1-\epsilon)$-approximate maximin. Consider an optimal maximin strategy. Round down the probability of each of its strategies to be in set P. Clearly, the updated probability of each strategy is more than a $(1-\epsilon)$ fraction of its original probability. Therefore, against every strategy of player 2, the updated strategy with probabilities in P obtains a payoff of at least $(1-\epsilon)\mathsf{OPT}$ concluding the proof. □

We use Lemma 116 to first fix the probabilities that are assigned to the strategies in the support and then construct them. We further need to fix the value of OPT a priori. This can be done via a binary search so long as by having probabilities

$p_1, \ldots, p_c$ and the value of OPT, we have an oracle that decides whether it is feasible to construct strategies $\mathbf{x}^1, \ldots, \mathbf{x}^c$ that guarantee an expected payoff of at least $(1-\epsilon)\mathsf{OPT}$ with these probabilities or not. Therefore it suffices for the continuous and discrete variants of Colonel Blotto to provide this oracle. This is our goal in the next two sections.

5.5.1 Continuous Colonel Blotto

Given probabilities $p_1, \ldots, p_c$ and the optimal maximin value OPT, our goal in this section is to construct c strategies $\mathbf{x}^1, \ldots, \mathbf{x}^c$ for the continuous variant of Colonel Blotto that guarantee a payoff of at least $(1-\epsilon)\mathsf{OPT}$ in expectation, against any strategy of player 2 (or report that this is infeasible). As in Section 5.1, we start by formulating the original problem as a (non-linear) program.

$$\begin{array}{lll} \text{find} & \mathbf{x}^1, \ldots, \mathbf{x}^c & \\ \text{subject to} & x_i^j \geq 0 & \forall i, j : i \in [k], j \in [c] \\ & \sum_{i \in [k]} x_i^j \leq n & \forall j \in [c] \\ & \sum_{j \in [c]} p_j \cdot \mathsf{u}_1(\mathbf{x}^j, \mathbf{y}) \geq \mathsf{OPT} & \forall \mathbf{y} \in \mathcal{S}_2 \end{array} \tag{5.19}$$

We need to better understand the last constraint of the formulation above to be able to solve it in polynomial time. For this, similar to the case of (u,p)-$\mathsf{maximin}$ strategies, we give an appropriately adapted definition of *critical tuples* and combine it with configurations that were introduced in Section 5.3.1.

Definition 117 (Critical tuples)**.** *Consider a tuple* $\mathbf{W} = (W_1, \ldots, W_k)$ *where each*

W_i *is a subset of* $[c]$. *We call* $\mathbf{W}$ *a* critical tuple *if and only if we have* $\sum_{i,j:j\in W_i} p_j w_i <$ OPT.

Recall from Section 5.3.1 that a configuration $\mathbf{G}$ is a vector of k matrices $G_1, \ldots, G_k$ which we call partial configurations, where for any $i \in [k]$, and for any $j_1, j_2 \in [c]$, the value of $G_i(j_1, j_2)$ is '$\leq$' if $x_i^{j_1} \leq x_i^{j_2}$ and it is '$\geq$' otherwise. Furthermore, for configuration $\mathbf{G}$ and critical tuple $\mathbf{W}$, define $z_i(\mathbf{G}, \mathbf{W}) := \arg\max_{j:j\notin W_i} x_i^j$. Note that it is crucial that $z_i(\mathbf{G}, \mathbf{W})$ is solely a function of $\mathbf{G}$ and $\mathbf{W}$ (and not the actual strategies $\mathbf{x}^1, \ldots, \mathbf{x}^c$) so long as strategies $\mathbf{x}^1, \ldots, \mathbf{x}^c$ comply with $\mathbf{G}$.

$$
\begin{array}{lll}
\text{find} & \mathbf{x}^1, \ldots, \mathbf{x}^c & \\
\text{subject to} & x_i^j \geq 0 & \forall i, j : i \in [k], j \in [c] \\
& \sum_{i\in[k]} x_i^j \leq n & \forall j \in [c] \\
& \text{ensure that } \mathbf{x}^1, \ldots, \mathbf{x}^c \text{ comply with } \mathbf{G} & \\
& \sum_{i\in[k]} x_i^{z_i(\mathbf{G},\mathbf{W})} > m & \text{for every critical tuple } \mathbf{W} = (W_1, \ldots, W_k)
\end{array}
\tag{5.20}
$$

This is essentially the same LP as LP 5.13 of Section 5.3.1 except that we use a different notion of critical tuples.

Observation 118. *Suppose that the configuration* $\mathbf{G}$ *of the optimal solution we seek to find is fixed. Then, the last constraint of LP 5.20 is equivalent to the last constraint of Program 5.19.*

Proof. Suppose at first that the last constraint of LP 5.20 is violated. We show that given that the solution should comply with $\mathbf{G}$, Program 5.19 is infeasible. To do so, consider the critical tuple $\mathbf{W} = (W_1, \ldots, W_k)$ for which we have $\sum_{i\in[k]} x_i^{z_i(\mathbf{G},\mathbf{W})} \leq m$.

Consider the strategy $\mathbf{y}$ of player 2 where $y_i = x_i^{z_i(\mathbf{G},\mathbf{W})}$. Clearly this is a feasible strategy for player 2 since it requires only $\sum_{i\in[k]} x_i^{z_i(\mathbf{G},\mathbf{W})}$ troops which is assumed to be no more than m. By Definition 117, the expected utility of any strategy of player 1 that complies with $\mathbf{G}$ is less than OPT against $\mathbf{y}$, meaning that Program 5.19 is infeasible.

Now suppose that LP 5.20 has a feasible solution $\mathbf{x}^1, \ldots, \mathbf{x}^c$. We show that this is also a valid solution for Program 5.19. Assume for the sake of contradiction that this is not true. That is, player 2 has a strategy $\mathbf{y}$ that prevents strategy $\mathbf{x}^1, \ldots, \mathbf{x}^c$ to obtain an expected payoff of OPT. For any $i \in [k]$, define $W_i := \{j : x_i^j > y_i\}$. Since the expected utility of player 1 by playing this strategy is less than OPT against $\mathbf{y}$, we have $\sum_{i,j:j\in W_i} p_j w_i < \mathsf{OPT}$. which means $(W_1, \ldots, W_k)$ is indeed a critical tuple. Consider the constraint of LP 5.20 corresponding to this critical tuple, we need to have $\sum_{i\in[k]} x_i^{z_i(\mathbf{G},\mathbf{W})} \leq m$ and therefore this constraint must be violated; contradicting the fact that $\mathbf{x}^1, \ldots, \mathbf{x}^c$ is a feasible solution of LP 5.20. □

Lemma 119. *LP 5.20 can be solved in polynomial time using the ellipsoid method.*

Proof. We only need to give a separating oracle for the last constraint of LP 5.20 since there are only polynomially many constraints of other types. Observation 118 shows that this constraint is essentially equivalent to the best response of player 2 which is used in Program 5.19. That is, if we can solve the best response of player 2 in polynomial time, we will be able to check whether any constraint of LP 5.20 is violated. We show that in fact, the best-response of player 2 can be solved in polynomial time. To do so, given c strategies $\mathbf{x}^1, \ldots, \mathbf{x}^c$ along with their

probabilities $p_1, \ldots, p_c$, we seek to find a strategy $\mathbf{y}$ of player 2 that maximizes his expected utility. For this, one can use a simple knapsack-like dynamic program $D(i, m')$ which essentially represents what expected payoff can be obtained from the first i battlefields given that player 2 uses only m' troops among them. One can easily confirm that this dynamic program can be updated by considering all possibilities of the number of troops over the ith battlefield and recurse over the prior battlefields. This gives a polynomial time algorithm for the best response of player 2, and therefore, a polynomial time algorithm for the separating oracle of LP 5.20. □

The lemma above shows that if we are given the actual configuration $\mathbf{G}$, we can solve the problem in polynomial time. In Sections 5.1.1 and 5.3.1, we showed how it suffices to only consider a polynomial number of configurations when all battlefields have the same weight (i.e., the uniform variant of the game). The same argument applies to the expected case since the players are still indifferent to the battlefields of the same weight. The generalization to the case of different battlefield weights follows from essentially the same approach described in Section 5.3.1. It suffices to consider the δ-uniform variant of the game for δ being a relatively smaller error threshold than ϵ. Then on each bucket of battlefields of the same weight, we only count the number of battlefields of each partial configuration instead of specifying the exact partial configuration of each battlefield. Using similar techniques as in Sections 5.1.1 and 5.3.1, it can be shown that it suffices to consider only a constant number of possibilities per bucket and therefore a polynomial number of configura-

tions in total. This concludes the continuous variant of the game when the objective is to compute an approximate maximin strategy.

Theorem 120. *For any $\epsilon > 0$, and any constant c, there exists a polynomial time algorithm to obtain a $(1-\epsilon)$-approximate maximin c-strategy for player 1 in the continuous Colonel Blotto game.*

5.5.2 Discrete Colonel Blotto

In this section we solve the same problem solved above for the discrete variant of Colonel Blotto. That is, given probabilities $p_1, \ldots, p_c$ and the optimal maximin value OPT, our goal is to construct c strategies $\mathbf{x}^1, \ldots, \mathbf{x}^c$ for the discrete variant of Colonel Blotto that guarantee a payoff of at least $(1-\epsilon)\mathsf{OPT}$ in expectation, against any strategy of player 2 (or report that this is infeasible).

We show that a similar approach to that of Section 5.4, with minor changes, can be applied to this case. Recall that the main idea that we used in Section 5.4 was to partition the battlefields into two disjoint subsets of heavy and light battlefields. Then roughly speaking, the idea was to perform an exhaustive search over the heavy battlefields and solve the problem over the light battlefields against a number of weaker adversaries each corresponding to a response of player 2 over the heavy battlefields. Fix δ to be a relatively smaller error threshold than ϵ. We say battlefield i is heavy if and only if $w_i \geq \delta\mathsf{OPT}$. We first show that w.l.o.g. we can assume that the number of heavy battlefields is at most $O(1)$ or otherwise a simple strategy obtains an expected utility of at least OPT.

Claim 121. *If the number of heavy battlefields is more than $2c^2/(\epsilon\delta)$, there exists an algorithm to find a c-strategy minimax strategy providing a utility of at least* OPT *in polynomial time if $m < (1-\epsilon)nc$.*

Proof. If the number of heavy battlefields is more than $2c^2/(\epsilon\delta)$, partition them into c disjoint subsets of size $2c/(\epsilon\delta)$. Consider the c-strategy of player 1 that chooses one of these subsets uniformly at random (i.e., with probability $1/c$) and distributes his troops among its battlefields almost uniformly (i.e., with pairwise difference of at most 1). Observe that even if in one of these strategies, player 1, wins c/δ heavy battlefields, the expected utility that he gets would be more than OPT, since

$$\frac{1}{c} \cdot \frac{c}{\delta} \cdot \delta \mathsf{OPT} = \mathsf{OPT}.$$

Therefore for player 2 to prevent player 1 from getting an expected utility of OPT, he has to win at least $(1-\epsilon)c/\epsilon\delta$ battlefields of each strategy which is not feasible for him since he needs to have $m \geq (1-\epsilon)nc$. □

Now, since we bound the number of heavy battlefields by a constant, we have our desired property that the number of relevant responses of player 2 over the heavy battlefields is bounded by a constant.

Observation 122. *Given a strategy of player 1 over the heavy battlefields, there exists a set S_2^h of a constant number of responses of player 2 over heavy battlefields among which an optimal response is guaranteed to exist.*

Proof. We can assume that the number of troops that player 2 puts on the ith heavy

battlefield is equal to the number of troops that player 1 puts in this battlefield in one of his c pure strategies. Therefore on each battlefield player 2 has $c+1$ options. Combined with the fact that the number of heavy battlefields is $O(1)$, this means there are only $(c+1)^{O(1)} \in O(1)$ relevant responses for player 2 over heavy battlefields. □

The observation above implies that the techniques of Section 5.4.3 where we perform an exhaustive search over the heavy battlefields and solve a dynamic program with as many dimensions as the number of responses of player 2 over the heavy battlefields is essentially feasible. It only remains to define the weaker adversary over the light battlefields. This turns out to be much simpler than the case of finding a (u,p)-maximin strategy.

The weaker adversary. Given strategies $\mathbf{x}^1, \ldots, \mathbf{x}^c$ with probabilities $p_1, \ldots, p_c$, for any $i \in [k]$ and any $j \in [c]$ define $u(i,j)$ to be the expected payoff that player 2 gets by putting exactly x_i^j troops in battlefield i. Moreover, we define the ratio $r(i,j)$ to be $u(i,j)/x_i^j$. The first action of the weaker adversary is to choose $i \in [k]$, $j \in [c]$ with maximum ratio $r(i,j)$ and put exactly x_i^j troops in battlefield i. Next, for any strategy j', we decrease $x_i^{j'}$ by x_i^j. Intuitively, this updates the additional number of troops that the weaker adversary has to put in this battlefield to win strategies with higher number of troops. Now over the updated strategies, the weaker adversary again computes the ratios, picks the one with the higher value and update the strategies accordingly. This continues until the weaker adversary spends all of his m troops.

Lemma 123. *By optimizing against the weaker adversary's greedy best response, one can guarantee an expected payoff of* $\mathsf{OPT} - w_{\max}$.

The lemma above shows that if we optimize our strategy against the weaker adversary over the light battlefields we obtain our desired $(1-\epsilon)\mathsf{OPT}$ expected utility so long as we allow the adversary to play any arbitrary strategy over the heavy battlefields. We capture the last iteration of algorithm above by *signatures* similar to Definition 102 of Section 5.4.3. Then by fixing strategy of player 1 over the heavy battlefields, and the signature of the weaker adversary for each of his constant relevant responses, we construct the optimal strategy of player 1 over the light battlefields in polynomial time using a dynamic program as in Section 5.4.3.

Theorem 124. *For any $\epsilon > 0$, and any constant c, there exists a polynomial time algorithm to obtain a $(1-\epsilon)$-approximate maximin c-strategy for player 1 in the discrete Colonel Blotto game if $n \geq (1+\epsilon)m/c$.*

Note that for the case where $n < m/c$, the second player can simply put $\max\{x_i^1, \ldots, x_i^c\}$ troops on battlefield i, ensuring that we get expected utility 0.

5.6 Further Complexity Results

For the purpose of studying its complexity, let us define Colonel Blotto as the following computational problem: Given a description of the discrete Colonel Blotto game — that is, the integer number of available troops for both players, and integer weights for the k battlefields — what are the maxmin strategies of the two players in the game whose utility is the probability of winning more than a threshold,

say half, of the total weight? Since the maxmin strategy is an exponential object, we only require the probability with which the last strategy (a specific allocation) is played[6]. It is clear that this problem can be solved by the ellipsoid algorithm in $2^{n^{O(1)}}$ (exponential) time, where n is the size of the input. We conjecture that the problem is exponential time-complete.

We have been unable to prove this conjecture; but as a promising start and consolation prize, we can show exponential time-completeness for the following generalization of the problem: In General Colonel Blotto[7] we are given a function $u_1 : \mathcal{S}_1 \times \mathcal{S}_2$ to the integers; that is, for each allocation of troops, u_1 computes the utility of Player 1 (the utility of Player 2 is, as always, its negation). The function u_1 is of course given as a Boolean circuit, U, since its explicit form is exponential. Thus, the circuit U is the input of the problem.

We can show the following:

Theorem 125. General Colonel Blotto *is exponential time-complete.*

Proof. We start with a problem we call Succinct Circuit Value: You are given a Boolean circuit with 2^n gates *implicitly* through another circuit C with n inputs. For each input $i \in [2^n]$, C outputs $2n+3$ bits interpreted as a triple $(t(i), j(i), k(i))$, where $t(i) \in \{0, 1, \vee, \wedge, \neg\}$ is the type of the gate, and $j(i), k(i) < i$ are the gates that are inputs of gate i. If $t(i) \in \{0, 1\}$ then $j(i) = k(i) = 0$, and if $t(i) = \neg, k(i) = 0$. The question asked is, does the output gate $2^n - 1$ evaluate to 1? It follows immediately

[6]It would be more natural to ask the value of the game; we require instead a component of the maxmin strategy for a technical reason: in our reduction below for the generalized problem, computing the value is easy: the game is symmetric, and the value is always zero.

[7]According to the Wikipedia, "General Colonel" is an extant rank in certain armies.

from the techniques in [80] that SUCCINCT CIRCUIT VALUE is exponential time-complete. Fo technical reasons, we require that not all inputs are zero (say, $t(0) = 1$), a restriction that obviously maintains complexity.

We reduce this problem to another we call SUCCINCT LINEAR INEQUALITIES: We are given a circuit C which, in input i, j gives the integer entry A_{ij} of an $M \times N$ matrix A — $j = N + 1$ it returns the value b_i of an M-vector b. The question is, does the system $Ax = b, x \geq 0$ have a solution? We claim that this problem is also exponential time-complete, by a simple reduction from SUCCINCT CIRCUIT VALUE, emulating the well known reduction between the non-succinct versions (see for example the textbook [81], page 222). In proof, we know from [80] that it suffices for such reductions between succinct problems to work that the corresponding reduction between the non-succinct problems is of a special kind called *projection*, and the vast majority of known reductions can be easily rendered as projections. Again for technical reasons, we modify slightly the construction by adding redundant constraints to the $0, 1, \neg$ gates to make sure that $N = 2^n$ and $M = 3 \cdot 2^n - 3$

Finally, we reduce SUCCINCT LINEAR INEQUALITIES to GENERAL COLONEL BLOTTO. For this part we follow the surprisingly recent reduction [82] from linear programming to zero-sum games (a 60 year old reduction due to Danzig was known to be incomplete, but it was far too much technical work to fix it...). Adler's reduction starts from a system $Ax = b, x \geq 0$ and produces a skew-symmetric

payoff matrix

$$P = \begin{bmatrix} 0 & 0 & A & e & -b \\ 0 & 0 & -e^T A & 1 & -e^T b \\ -A^T & A^T e & 0 & 0 & 1 \\ -e^T & -1 & 0 & 0 & 1 \\ b^T & -b^T e & 0 & -1 & 0 \end{bmatrix}.$$

Actually, Adler starts by adding one extra row to A to ensure that $Ax = 0$ has no nonzero, nonnegative solutions; however, in our case this is guaranteed by requiring that the original circuit has at least one nonzero input. He shows that the system $Ax = b, x \geq 0$ has a solution iff the last component of the (symmetric) maxmin strategy of this zero-sum game is nonzero.

What remains is to label the $L = M + N + 3 = 4^n$ rows and columns of this matrix by allocations of troops by the two general colonels, and define the utility u_1. The number of battlefields is $4n$, and we define the set of *feasible* allocations to be of the form $C(S)$ where S is any subset of the first half of the battlefields. $C(S)$ assigns one troop to each battlefield in S, and each battlefield j such that $j - 2n \notin S$. That is, troop assignments in the first and the second half of the battlefields complement each other. Finally, we define the utility function u_1: Given two allocations A, B, $u_1(A, B)$ is defined as follows:

- If both A and B are infeasible, $u_1(A, B) = 0$;
- if A is feasible and B is infeasible, $u_1(A, B) = -c$, where c is larger than any payoff; this way, player 1 is disincentivized from using A;

- similarly, if B is feasible and A is infeasible, $u_1(A, B) = c$;
- finally, if both A and B are feasible, say $A = C(S)$ and $B = C(T)$ for subsets S, T of $[2n]$, $u_1(A, B) = P_{ij}$, the (i, j)-th entry of the payoff matrix P constructed in Adler's reduction, where the binary representation of i is the set S followed by the set $[2n] - S$, and similarly for j and T.

It is clear that infeasible allocations are dominated, and can thus be eliminated from the game. The feasible strategies are in one-to-one correspondence with the rows and columns of the matrix P, and thus the maxmin of the General Colonel Blotto game is the same as the maxmin of P. Therefore, the last component of the maxmin strategy is nonzero if and only if the original circuit has value one, and the reduction is complete. □

Bibliography

[1] Mahdi Ahmadinejad, Sina Dehghani, MohammadTaghi Hajiaghayi, Brendan Lucier, Hamid Mahini, and Saeed Seddighin. From duels to battefields: Computing equilibria of blotto and other games. *AAAI*, 2016.

[2] Soheil Behnezhad, Sina Dehghani, Mahsa Derakhshan, MohammadTaghi HajiAghayi, and Saeed Seddighin. Faster and simpler algorithm for optimal strategies of blotto game. *AAAI*, 2017.

[3] Soheil Behnezhad, Avrim Blum, Mahsa Derakhshan, MohammadTaghi HajiAghayi, Mohammad Mahdian, Christos H Papadimitriou, Ronald L Rivest, Saeed Seddighin, and Philip B Stark. From battlefields to elections: Winning strategies of blotto and auditing games. In *SODA*, 2018.

[4] Soheil Behnezhad, Mahsa Derakhshan, Mohammadtaghi Hajiaghayi, and Saeed Seddighin. Spatio-temporal games beyond one dimension. In *ACM EC*, 2018.

[5] Soheil Behnezhad, Avrim Blum, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Christos H Papadimitriou, and Saeed Seddighin. Optimal strategies of blotto games: Beyond convexity. *ACM EC*, 2019.

[6] John Nash. Non-cooperative games. *Annals of mathematics*, 1951.

[7] Constantinos Daskalakis and Christos H Papadimitriou. Three-player games are hard. In *ECCC*, 2005.

[8] Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player nash equilibria. *JACM*, 2009.

[9] Constantinos Daskalakis, Paul W Goldberg, and Christos H Papadimitriou. The complexity of computing a nash equilibrium. *SICOMP*, 2009.

[10] Paul W Goldberg and Christos H Papadimitriou. Reducibility among equilibrium problems. In *STOC*, 2006.

[11] Xi Chen and Xiaotie Deng. Settling the complexity of two-player nash equilibrium. In *FOCS*, 2006.

[12] Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Computing nash equilibria: Approximation and smoothed complexity. In *FOCS*, 2006.

[13] George B Dantzig. *Linear programming and extensions*. 1963.

[14] Jugal Garg, Albert Xin Jiang, and Ruta Mehta. Bilinear games: Polynomial time algorithms for rank based subclasses. In *Internet and Network Economics*. 2011.

[15] Spyros Kontogiannis and Paul Spirakis. Exploiting concavity in bimatrix games: New polynomially tractable subclasses. In *ARCO*. 2010.

[16] Richard J Lipton, Evangelos Markakis, and Aranyak Mehta. Playing large games using simple strategies. In *ACM EC*, 2003.

[17] Émile Borel. La théorie du jeu et les équations intégrales à noyau symétrique. *Comptes Rendus de l'Académie*, 1921.

[18] Émile Borel. The theory of play and integral equations with skew symmetric kernels. *Econometrica*, 1953.

[19] M. Fréchet. Commentary on the three notes of emile borel. *Econometrica*, 1953.

[20] M. Fréchet. Emile borel, initiator of the theory of psychological games and its application. *Econometrica*, 1953.

[21] J. von Neumann. Communication on the borel notes. *Econometrica*, 1953.

[22] Roger B Myerson. Incentives to cultivate favored minorities under alternative electoral systems. *American Political Science Review*, 1993.

[23] Jean-Francois Laslier and Nathalie Picard. Distributive politics and electoral competition. *Journal of Economic Theory*, 2002.

[24] Jennifer Merolla, Michael Munger, and Michael Tofias. In play: A commentary on strategies in the 2004 us presidential election. *Public Choice*, 2005.

[25] Subhasish M Chowdhury, Dan Kovenock, and Roman M Sheremeta. An experimental investigation of colonel blotto games. *Economic Theory*, 2009.

[26] Dan Kovenock and Brian Roberson. Conflicts with multiple battlefields. Technical report, 2010.

[27] Dan Kovenock and Brian Roberson. Coalitional colonel blotto games with application to the economics of alliances. *Journal of Public Economic Theory*, 2012.

[28] John W Tukey. A problem of strategy. *Econometrica*, 1949.

[29] Donald W Blackett. Some blotto games. *Naval Research Logistics Quarterly*, 1954.

[30] Donald W Blackett. Pure strategy solutions to blotto games. *Naval Research Logistics Quarterly*, 1958.

[31] Richard Bellman. On colonel blotto and analogous games. *Siam Review*, 1969.

[32] Martin Shubik and Robert James Weber. Systems defense games: Colonel blotto, command and control. *Naval Research Logistics Quarterly*, 1981.

[33] Jonathan Weinstein. Two notes on the blotto game. *Manuscript, Northwestern University*, 2005.

[34] Brian Roberson. The colonel blotto game. *Economic Theory*, 2006.

[35] Dmitriy Kvasov. Contests with limited resources. *Journal of Economic Theory*, 2007.

[36] Sergiu Hart. Discrete colonel blotto and general lotto games, 2007.

[37] Russell Golman and Scott E Page. General blotto: games of allocative strategic mismatch. *Public Choice*, 2009.

[38] NSF. Umd-led team first to solve well-known game theory scenario, 2016.

[39] DARPA. Umd-led team first to solve well-known game theory scenario, 2016.

[40] Business Insider. Scientists say they can predict two-party outcomes after solving a 95-year-old game theory problem, 2016.

[41] Europapress. El juego del coronel blotto. un algoritmo indica la mejor estrategia, 2016.

[42] tivi. Kenest presidentti? tietokone tiet, 2016.

[43] Forskning. Forskere vil regne ut hvem som blir usas neste president, 2016.

[44] Mandiner. Jtkelmlettel nyerjen elnkvlasztst!, 2016.

[45] H Bernhard, BH Korte, and J Vygen. *Combinatorial optimization: Theory and algorithms*. 2008.

[46] Thomas Rothvoß. The matching polytope has exponential extension complexity. In *STOC*, 2014.

[47] David Applegate and Edith Cohen. Making intra-domain routing robust to changing and uncertain traffic demands: understanding fundamental tradeoffs. In *ATAP*, 2003.

[48] Richard J Lipton, Evangelos Markakis, and Aranyak Mehta. Playing large games using simple strategies. In *ACM EC*, 2003.

[49] Herbert Alexander Simon. *Models of bounded rationality: Empirically grounded economic reason*. 1982.

[50] Ariel Rubinstein. *Modeling bounded rationality*. 1998.

[51] Itzhak Gilboa and Eitan Zemel. Nash and correlated equilibria: Some complexity considerations. *Games and Economic Behavior*, 1989.

[52] Vladimir Estivill-Castro and Mahdi Parsa. Computing nash equilibria gets harder: new results show hardness even for parameterized complexity. In *ASOC*, 2009.

[53] Nicole Immorlica, Adam Tauman Kalai, Brendan Lucier, Ankur Moitra, Andrew Postlewaite, and Moshe Tennenholtz. Dueling algorithms. In *STOC*, 2011.

[54] Soheil Behnezhad, Sina Dehghani, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, and Saeed Seddighin. Faster and simpler algorithm for optimal strategies of blotto game. In *AAAI*, 2017.

[55] Maurice Sion. *General Minimax Theorems*. 1957.

[56] Yossi Azar, Edith Cohen, Amos Fiat, Haim Kaplan, and Harald Racke. Optimal oblivious routing in polynomial time. In *STOC*, 2003.

[57] Martin Grótschel, Lacuteászló Lovász, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*. 1988.

[58] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. 1998.

[59] Martin Grötschel, Lacuteászló Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1981.

[60] Nicole Immorlica, Adam Tauman Kalai, Brendan Lucier, Ankur Moitra, Andrew Postlewaite, and Moshe Tennenholtz. Dueling algorithms. In *STOC*, 2011.

[61] Sina Dehghani, Mohammad Taghi Hajiaghayi, Hamid Mahini, and Saeed Seddighin. Price of Competition and Dueling Games. In *ICALP*, 2016.

[62] Thomas Rothvoss. The matching polytope has exponential extension complexity. In *STOC*, 2014.

[63] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*. 1994.

[64] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V Vazirani. *Algorithmic game theory*. 2007.

[65] Jack Edmonds. Matroids and the greedy algorithm. *Mathematical programming*, 1971.

[66] R Kipp Martin. Using separation algorithms to generate mixed integer model reformulations. *Operations Research Letters*, 1991.

[67] Michel X Goemans. Smallest compact formulation for the permutahedron. *Mathematical Programming*, 2015.

[68] Thomas Rothvoß. Some 0/1 polytopes need exponential size extended formulations. *Mathematical Programming*, 2013.

[69] Mihalis Yannakakis. Expressing combinatorial optimization problems by linear programs. In *STOC*, 1988.

[70] T.H. Cormen. *Introduction to Algorithms*. 2009.

[71] M.S. Bazaraa, J.J. Jarvis, and H.D. Sherali. *Linear Programming and Network Flows*. 2011.

[72] Uriel Feige, Kamal Jain, Mohammad Mahdian, and Vahab Mirrokni. Robust Combinatorial Optimization with Exponential Scenarios. *Integer programming and combinatorial optimization*, 2007.

[73] MohammadHossein Bateni, Sina Dehghani, MohammadTaghi Hajiaghayi, and Saeed Seddighin. Revenue maximization for selling multiple correlated items. In *ESA*. 2015.

[74] Xinye Li and Andrew Chi-Chih Yao. On Revenue Maximization for Selling Multiple Independently Distributed Items. *NAS*, 2013.

[75] Moshe Babaioff, Nicole Immorlica, Brendan Lucier, and S Matthew Weinberg. A Simple and Approximately Optimal Mechanism for an Additive Buyer. In *FOCS*, 2014.

[76] Aviad Rubinstein and S Matthew Weinberg. Simple Mechanisms for a Subadditive Buyer and Applications to Revenue Monotonicity. In *ACM EC*, 2015.

[77] Jugal Garg, Albert Xin Jiang, and Ruta Mehta. Bilinear games: Polynomial time algorithms for rank based subclasses. In *International Workshop on Internet and Network Economics*, 2011.

[78] Haifeng Xu. The Mysteries of Security Games: Equilibrium Computation Becomes Combinatorial Algorithm Design. In *ACM EC*, 2016.

[79] Sinong Wang and Ness Shroff. Security Game with Non-additive Utilities and Multiple Attacker Resources. *arXiv preprint arXiv:1701.08644*, 2017.

[80] Christos H Papadimitriou and Mihalis Yannakakis. A note on succinct representations of graphs. *Information and Control*, 1986.

[81] Sanjoy Dasgupta, Christos H Papadimitriou, and Umesh Vazirani. *Algorithms*. 2006.

[82] Ilan Adler. The equivalence of linear programs and zero-sum games. *International Journal of Game Theory*, 2013.