DESIGN IS AS EASY AS OPTIMIZATION*

DEEPARNAB CHAKRABARTY†, ARANYAK MEHTA‡, AND VIJAY V. VAZIRANI§

Abstract. We consider the class of max-min and min-max optimization problems subject to a global budget constraint. We undertake a systematic algorithmic and complexity-theoretic study of such problems, which we call problems design problems. Every optimization problem leads to a natural design problem.

Our main result uses techniques of Freund-Schapire [FS99] from learning theory, and its generalizations, to show that for a large class of optimization problems, the design version is as easy as the optimization version.

We also observe the relationship between max-min design problems and fractional packing problems. In particular, we obtain in a systematic fashion results about the fractional packing number of Steiner trees.

1. Introduction. In this paper, we undertake a systematic study of max-min and min-max optimization problems subject to a global budget constraint. We call such problems *design problems*. Every optimization problem leads to a natural design problem; if the optimization problem is a minimization (maximization) problem, its design version is a max-min (min-max) problem.

The process of obtaining a design problem from an optimization problem is formally defined in Section 2. As an illustration, the design problem obtained from the sparsest cut problem is the following: given an undirected graph $G = (V, E)$ and a bound B on the total weight, find a distribution of the weight B on the edges of G so that the sparsity of the sparsest cut is maximized. Observe that this is a max-min problem.

The history of such problems goes back all the way to Fulkerson [Ful59], who considered the problem of maximizing the minimum cut in a network whose edge capacities could be augmented, given a bound on the total augmentation allowed. Observe that since the minimum cut in a network equals the maximum flow, this max-min problem can be transformed into a pure maximization problem: that of finding the augmented network that supports maximum possible flow. Frederickson and Solis-Oba [FSO99] considered the problem of augmenting the weights of an undirected graph to maximize the weight of the minimum spanning tree, given a bound on the total augmentation allowed and gave a strongly polynomial time algorithm for the same. They later [FSO97] generalize the problem to maximize the minimum weight basis of a matroid.

Jüttner [Jüt06] studied a class of max-min and min-max problems, which he called *budgeted optimization problems*, which generalizes the above two problems, and showed a general transformation to underlying optimization problems. Start with any optimization problem for which the set of feasible solutions forms a polytope. Then, the max-min or min-max problem obtained from it (depending on whether the original problem is a minimization or maximization problem) is in this class.

Jüttner showed the general result that if the original optimization problem has a strongly polynomial algorithm, then so does the budgeted optimization problem. A key step in obtaining this result is captured in the solution to Fulkerson's above stated problem. Without loss of generality, assume that the budgeted optimization problem is a max-min problem, which has been obtained from a minimization problem. Now, using fact that the set of feasible solutions of the latter form a polytope, it can be written as a minimization LP. Its dual, a maximization LP, also achieves the same optimal solution, thereby transforming the max-min problem into a max-max problem, which is simply a maximization problem. The latter is solved using Megiddo's parametric search method [Meg79].

In retrospect, Jüttner has carved out a subclass of design problems that, via a polynomial amount of work, can be restated as optimization problems. Here we study a more general class of problems in which the underlying optimization problem can have an arbitrary set of feasible solutions, may not

*Parts of this work appeared in the Proceedings of International Colloquium on Automata, Language and Programs (ICALP), 2006 [CMV06]

†Department of Computer and Information Sciences, University of Pennsylvania (deeparnab@gmail.com).

‡Google, Inc., Mountain View, USA, (aranyak@google.com).

§College of Computing, Georgia Tech, Atlanta GA 30332, (vazirani@cc.gatech.edu). Supported by NSF Grant CCF-0728640 and ONR Grant N000140910755.

even be polynomial time solvable and moreover may not even be a linear (but needs to be a concave minimization problem or a convex maximization problem; see Section 2). On the negative side, we only give polynomial, and not strongly polynomial algorithms (exact or approximation) for design problems, whenever the optimization problem has a polynomial time (exact or approximation) algorithm.

This brings us to the justification of the name "design problem": Assume that the underlying optimization problem is a minimization problem, i.e., among the set of feasible solutions, which is typically exponentially large, it is seeking a minimum cost solution. Then the corresponding design problem, with a given budget, is seeking an instance (among all instances satisfying the budget constraint) in which the minimum cost solution is as large as possible. Thus the task at hand is to design the best *instance* satisfying certain constraints and properties. With this explanation, it should be clear that design problems arise in numerous applications. For instance, consider the problem of distributing a fixed amount of capacity across the edges of a network so as to maximize the amount of concurrent flow that can be sustained between any two pairs of nodes - this is just the design version of the sparsest cut problem.

Two prominent design problems studied recently are: Boyd, Diaconis and Xiao [BDX04] study the design of the fastest mixing Markov chain on a graph with a budget constraint on the weights of the edges of a fixed graph. Elson, Karp, Papadimitriou and Shenker [EKPS04] study the synchronization design problem in sensornets, which is essentially the problem of finding a Markov chain on a graph that minimizes the maximum commute time. It is instructive to note that neither of these design problems is a budgeted optimization problem.

1.1. Overview of results. Our main result is that for a large class of optimization problems, the design version of the problem is as easy to solve as the optimization problem itself. The class of problems we show this for are minimization problems with concave objective functions, and maximizations problems with convex objective functions. An important special class is the class of problems with linear objective functions. These classes will be defined formally in Section 2. We state our results here for minimization problems - the maximization versions have analogous results.

- In Section 3.1, we observe that for a minimization problem Π with a concave objective function, the corresponding maxmin design problem $D(\Pi)$ can be set up as a convex optimization problem. Moreover, if we use the ellipsoid method to solve the problem, then the separation oracle required is Π itself. Thus, if we can solve Π in polynomial time, then we can also solve $D(\Pi)$ in polynomial time. Furthermore, if Π has an α-factor approximation algorithm, then using the ellipsoid method along with a binary search we can get an α approximation for $D(\Pi)$ as well.
- In Section 3.2 we include, for completeness, the observation from [Jüt06] that if the optimization problem Π can be set up as a linear program, then the design problem $D(\Pi)$ can be set up as another similar linear program. If Π itself cannot be set up as a linear program, but there is a linear program whose solution is within a factor of α of the optimal solution of Π (e.g., an LP relaxation of an integer program for Π), then we can find a linear program for $D(\Pi)$ which has an optimal solution within a factor α of the optimal solution to the design problem.
- In Section 4, we give the main algorithmic result of this paper – a second general method for solving the design problem. This method is much more efficient than the ellipsoid method of Section 3.1. We set up the design problem $D(\Pi)$ of an optimization problem Π as a two player zero-sum game and show that the $D(\Pi)$ seeks the minmax value of this game. We apply the adaptive learning techniques of Freund-Schapire [FS99] in the linear case and that of Flaxman et.al [FKM05] in the concave case to solve the game. This results in an iterative scheme to solve $D(\Pi)$ within an additive error ϵ by using the approximation algorithm for Π only $O(\ln n/\epsilon^2)$ times. If the algorithm for Π has a worst case factor of α, then we solve $D(\Pi)$ up to a factor of α with an additional ϵ additive error. We note that these algorithms are randomized algorithms.
- In Section 5 we investigate the relationship between the complexity of an optimization problem and its corresponding design problem. We already establish in Section 3.1 that if a linear optimization problem is in **P** then so is its design version. We provide an example in which a

linear optimization problem is **NP**-complete but its design version is in **P**, and another example in which a linear optimization problem and its design version are **NP**-hard.

- In Section 6, using the observation that for any minimization problem, the max-min design optimum with budget 1 is the reciprocal of the fractional packing number, we obtain different results about the fractional packing number of Steiner trees (Section 6.1) using the different LP relaxations for the minimum Steiner tree problem. For instance, one such result is that the fractional packing number of *spanning trees* equals the strength of a graph which follows from the famous result of Nash-Williams [NW61] and Tutte [Tut61] about integral packing number of spanning trees. Another result is that of Agarwal and Charikar [AC04] which connects the network coding gain in an undirected graph with the integrality gap of the bidirected cut relaxation.

2. Problem Definition. We present a general framework to define the design versions of optimization problems:

Definition 1 An ***optimization problem*** Π consists of a set of *valid instances* $\mathcal{I}_\Pi$. Each instance I is a tuple $(E_I, \mathcal{S}_I, \mathbf{w}_I)$. Henceforth we will drop the subscript when the instance is clear from context. E is a universe of *elements*, and each element $e \in E$ has an associated weight $\mathbf{w}(e) \geq 0$, a rational number, giving the vector $\mathbf{w}$. Throughout we will let $n = |E|$. Each instance also has a set of *feasible solutions* [1] $\mathcal{S}$. For an instance $I = (E, \mathcal{S}, \mathbf{w})$, and a feasible solution $S \in \mathcal{S}$, the value of the objective function is a function of S and $\mathbf{w}$, denoted as $f_S(\mathbf{w})$. For a minimization problem, the goal is to find an optimal solution:

$$S^* = \operatorname*{argmin}_{S \in \mathcal{S}} f_S(\mathbf{w})$$

We also define:

$$OPT_\Pi((E, \mathcal{S}, \mathbf{w})) = \min_{S \in \mathcal{S}} f_S(\mathbf{w})$$

For $\alpha \geq 1$, a feasible solution S' is called an α-approximate solution to I if:

$$f_{S'}(\mathbf{w}) \leq \alpha \cdot OPT_\Pi(I)$$

An algorithm is called an α-approximation algorithm for the problem Π if for every instance I of Π, the algorithm returns an α-approximate solution to I. The goal of a maximization problem is defined similarly.

Definition 2 The ***maxmin design version*** $D(\Pi)$ of a minimization problem Π is defined as follows: For every collection of valid instances of Π of the form $I = (E_I, \mathcal{S}_I, \cdot)$, there is one valid instance of $D(\Pi)$: $J = (E_J, \mathcal{S}_J, B_J)$, where $E_J = E_I$, $\mathcal{S}_J = \mathcal{S}_I$, and B_J is a rational number, called the weight budget. A feasible solution to J is a weight vector $\mathbf{w} = (\mathbf{w}(e))_{\{e \in E_J\}}$, which satisfies the *budget constraint* $\sum_{e \in E_J} \mathbf{w}(e) \leq B_J$. Every feasible solution $\mathbf{w}$ to J leads to an instance $I = (E_I, \mathcal{S}_I, \mathbf{w})$ of the optimization problem Π.

The goal of the maxmin design problem is to find a feasible solution $\mathbf{w}$ so that the minimum objective function value of the resulting instance of the minimization problem is as large as possible. That is, the goal is to find an optimal solution:

$$\mathbf{w}^* = \operatorname*{argmax}_{\mathbf{w}: \sum_e \mathbf{w}(e) \leq B} OPT_\Pi((E, \mathcal{S}, \mathbf{w}))$$

We also define:

$$OPT_{D(\Pi)}((E, \mathcal{S}, B)) = \max_{\mathbf{w}: \sum_e \mathbf{w}(e) \leq B} OPT_\Pi((E, \mathcal{S}, \mathbf{w}))$$

[1] The number of feasible solutions may be, and usually is, exponential in $n = |E|$

For $\alpha \geq 1$, a weight vector $\mathbf{w}'$ is called an α-approximate solution to I if:

$$OPT_{\Pi}((E, \mathcal{S}, \mathbf{w}')) \geq \frac{1}{\alpha} \cdot OPT_{D(\Pi)}$$

An algorithm is called an α-approximation algorithm for a design problem $D(\Pi)$ if for every instance I of $D(\Pi)$, the algorithm returns an α-approximate solution to I.

The minmax design version of a maximization problem is defined similarly.

Definition 3 An optimization problem Π (and its design version $D(\Pi)$) is called ***linear*** if all its instances $I = (E, \mathcal{S}, \mathbf{w})$, are of the following form: $\mathcal{S} \subseteq 2^E$, and $\forall\ S \in \mathcal{S}:\ \ f_S(\mathbf{w}) = \sum_{e \in S} a_{e,S} \mathbf{w}_e$, for some $a_{e,S} \geq 0$. A more general class of problems has the functions f_S being convex or concave functions of $\mathbf{w}$. We shall call these ***concave minimization*** and ***convex maximization*** problems.

Examples: Most optimization problems on graphs are linear, as defined above. For example, in the Minimum Spanning Tree problem (Traveling-Salesman, Sparsest Cut), an instance $I = (E, \mathcal{S}, \mathbf{w})$ has E being the set of edges of the given graph, $\mathcal{S}$ being the collection of all sets of edges which form spanning trees (Hamiltonian cycles, cuts), and $\mathbf{w}$ being the given weights on the edges. An instance $(E, \mathcal{S}, B)$ of the design version of these problems would be to allocate a budget of B to the edges of the graph so as to maximize the weight of the minimum weight spanning tree (maximize the weight of the best TSP tour, make the sparsest cut as dense as possible). Clearly, some design problems make more intuitive sense than others.

An example of a convex maximization problem is that of finding the maximum commute time of a random walk on a graph over different pairs of vertices. Here an instance is $I = (E, \mathcal{S}, \mathbf{w})$, where E is the set of edges of the graph, $\mathcal{S}$ is the collection of pairs of vertices, and the w_es are the relative conductances of the edges, giving the transition probabilities. The functions f_S are the commute time functions, known to be convex (see Section 4.2.1 for details). The design version of this problem is that of assigning transition probabilities to minimize the maximum commute time.

3. Solving design problems.

3.1. A general technique based on the ellipsoid method. Consider a concave minimization problem and its corresponding max-min design problem. The analysis for convex min-max design problems is similar. The following theorem states that if the optimization version can be solved in polynomial time, then the design version can be solved up to additive error ϵ using the ellipsoid method.

Theorem 3.1
If we have an algorithm which solves the minimization problem $\Pi = (E, \mathcal{S}, \mathbf{w})$ with a concave objective function $f_S(\mathbf{w})$ in polynomial time, then for any $\epsilon > 0$, we can solve the corresponding max-min design problem $D(\Pi)$ up to an additive error of ϵ in time polynomial in n and $\log \frac{1}{\epsilon}$.

Proof. Note that the value of the optimal max-min design is given by the following program:

$$OPT_{D(\Pi)} := \max\{\lambda :\ \lambda - f_S(\mathbf{w}) \leq 0,\ \forall S \in \mathcal{S};\quad \sum_{e \in E} \mathbf{w}(e) \leq B;\quad \mathbf{w}(e) \geq 0,\ \forall e \in E\} \tag{3.1}$$

If $f_S()$ is concave, then the above program is convex, since for any two feasible solutions $(\lambda, \mathbf{w})$ and $(\lambda', \mathbf{w}')$ and $0 \leq \mu \leq 1$, we have

$$f_S(\mu \mathbf{w} + (1-\mu)\mathbf{w}')\ \ \geq\ \ \mu f_S(\mathbf{w}) + (1-\mu) f_S(\mathbf{w}')\ \ \geq\ \ \mu\lambda + (1-\mu)\lambda'$$

Hence, one can solve the above program using the ellipsoid method. Assuming an upper bound F on the optimum, the algorithm proceeds with a guess λ of the optimum to the program and tests for emptiness of the convex feasible set. For any $\epsilon > 0$, in time polynomial in the input size and $\log \frac{1}{\epsilon}$, the ellipsoid method (see [GLS88]) using the minimization problem as a separating oracle[2] to construct the

[2]Here, and throughout, we will say that an (approximation) algorithm solves a optimization problem if it gives the (approximately) optimum value as well as a set S which achieves this (approximately) optimum value.

ellipsoids, returns a feasible $\mathbf{w}$, or asserts that optimum is smaller than $\lambda + \epsilon$. Via a binary search to find λ^* which takes time $\log F$, the theorem follows by noting that an upper bound on λ^* is polynomial in the size of the value returned by the minimization problem. □

In the next theorem we show if the minimization problem has an α-approximation, then so does the max-min design version. The technique, a now standard trick of designing an α-approximate separation oracle, first appeared in the work of Carr and Vempala [CV02]. We include it for completeness.

Theorem 3.2
If we have a polynomial time algorithm returning an α-approximation to the optimization problem Π, then we can find, for any $\epsilon > 0$, an approximation algorithm for the design problem $D(\Pi)$, with a multiplicative factor of α and an additive error of ϵ.

Proof. We have a polytime algorithm which, given $(E, \mathcal{S}, \mathbf{w})$, returns a set S with objective function value guaranteed to be at most α-factor away from the actual optimum: $f_S(\mathbf{w}) \leq \alpha \min_{T \in \mathcal{S}} f_T(\mathbf{w})$. As in the proof of Theorem 3.1, given a guess λ, we run ellipsoid to check if there exists a feasible $\mathbf{w}$. The difference now is that the separation oracle is the approximate minimization algorithm. Thus, for any $\epsilon > 0$, the ellipsoid algorithm returns, in time polynomial in input and $\log \frac{1}{\epsilon}$, a solution $(\lambda, \mathbf{w})$, so that the optimum is less than $\lambda + \epsilon$. However, since the separation oracle is approximate, $(\lambda, \mathbf{w})$ might not be feasible itself. Nevertheless, by the guarantee of the approximation, we know $(\lambda/\alpha, \mathbf{w})$ is feasible, which implies the theorem. □

Remark 3.1 We note that for linear optimization problems where the function $f_S()$ is linear, the program (3.1) is a linear program, and (see [GLS88]) the above two theorems hold without any additive error.

The ellipsoid method may need to take a number of steps equal to a large polynomial. In each step we need to solve an instance of the optimization problem Π. The ellipsoid method also takes a huge time in practice. This motivates us to look for faster algorithms for the design problem. In Section 4, we will provide a different general method which works much faster.

3.2. A technique based on LP-relaxation. In this section we describe a general technique for solving design problems, in the case that we have a linear programming relaxation for the minimization problem Π. This technique has been described in [Jüt06], and we include it here only for the sake of completeness.

Suppose we have:

$$OPT_\Pi \geq \min \{ \quad \mathbf{w} \cdot \mathbf{x} \quad \text{s.t} \quad A\mathbf{x} \geq \mathbf{b}; \quad \mathbf{x} \geq \mathbf{0} \quad \} \tag{3.2}$$

Moreover, suppose there is an α-approximate polynomial time algorithm which returns a solution S with $f_S(\mathbf{w}) \leq \alpha \cdot L \leq \alpha \cdot OPT_\Pi$, where L is the solution to LP(3.2) (That is, the integrality gap of the LP is at most α). Then we have an α-approximation for the design version as well.

Theorem 3.3
If we have an LP relaxation for the optimization problem Π, and a polynomial time algorithm producing a solution within $\alpha \geq 1$ times the LP optimum, then we can produce an α approximation algorithm for the corresponding design problem $D(\Pi)$ which requires solving a single LP having one constraint more than that of the LP relaxation.

Proof. Look at the dual of LP(3.2).

$$\max \{ \quad \mathbf{b} \cdot \mathbf{y} \quad \text{s.t} \quad \mathbf{y}^T A \leq \mathbf{w}; \quad \mathbf{y} \geq \mathbf{0} \quad \} \tag{3.3}$$

In the design problem, note that the weight vector $\mathbf{w}$ is no longer in the objective function but appears in the constraints. Parameterizing the program on $\mathbf{w}$, let the optimal solution to (3.3) be $D(\mathbf{w})$. From the previous supposition, we know there is an algorithm giving a set S with the guarantee, $D(\mathbf{w}) \leq f_S(\mathbf{w}) \leq \alpha D(\mathbf{w})$ for all weight vectors $\mathbf{w}$.

To solve the design problem, we consider $\mathbf{w}$ as a variable in(3.3), and add the constraint that the total weight is bounded by B. Thus we solve the following LP

$$\max \{ \quad \mathbf{b} \cdot \mathbf{y} \quad \text{s.t} \quad \mathbf{y}^T A - \mathbf{w} \leq \mathbf{0}; \quad \mathbf{w} \cdot \mathbf{1} \leq B; \quad \mathbf{y}, \mathbf{w} \geq \mathbf{0} \quad \} \tag{3.4}$$

Let the optimal solution to (3.4) be D^*. Let $\mathbf{w}'$ be the optimum vector returned in the solution of (3.4). Note that for any weight vector $\mathbf{w}$ satisfying $\mathbf{w} \cdot \mathbf{1} \leq B$, we have $D(\mathbf{w}) \leq D^*$ with equality at $\mathbf{w}'$. Solve (3.2) with $\mathbf{w}'$ and obtain a set T with the guarantee $D^* \leq f_T(\mathbf{w}') \leq \alpha D^*$.

We now claim that $T, \mathbf{w}'$ gives an α approximation to the design problem. To see this, suppose $\mathbf{w}^*$ was the weight vector achieving the maxmin design. Moreover, suppose S was the set that minimized its objective value given $\mathbf{w}^*$. We need to show $\alpha f_T(\mathbf{w}') \geq f_S(\mathbf{w}^*)$. To see this note $f_S(\mathbf{w}^*) \leq \alpha D(\mathbf{w}^*) \leq \alpha D^* \leq \alpha f_T(\mathbf{w}')$. □

As a corollary we get a $\log n$ approximation to maximum min-multicut, a 2-approximation to the maximum min weighted vertex cover, a 2-approximation for max-min Steiner trees and many such problems which have approximation algorithms via LP-relaxations.

4. Faster algorithms for Design Problems. In this section we provide a general method to solve design problems. In the case of linear optimization and design problems this method works much faster than the method in Section 3.1 (calling the optimization algorithm only $O(\log n)$ times rather than $poly(n)$ times) but provides a weaker approximation (runs in time poly$(1/\epsilon)$, rather than poly$(\log 1/\epsilon)$). In Section 4.1 we consider the conceptually simpler case of design versions of linear optimization problems, before moving on to the more general concave minimization and convex maximization problems in Section 4.2.

4.1. Linear Design Problems, Zero-sum Games and Multiplicative Updates. Recall the definition of linear optimization problems and their design versions: the instances $I = (E, \mathcal{S}, \mathbf{w})$, are of the form $\mathcal{S} \subseteq 2^E$, and $\forall\ S \in \mathcal{S}:\ \ f_S(\mathbf{w}) = \sum_{e \in S} a_{e,S} \mathbf{w}_e$. In this section we shall take all the $a_{e,S} = 1$ for the sake of succinct notation – all the proofs extend naturally to the general case – so that $f_S(\mathbf{w}) = \sum_{e \in S} \mathbf{w}_e$.

Definition 4 Given an instance $I = (E, \mathcal{S}, B{=}1)$ of a maxmin design problem $D(\Pi)$, the **equivalent zero-sum game** $G(I)$ is defined by an $|E| \times |\mathcal{S}|$ matrix as follows: the rows are indexed by E and the columns by $\mathcal{S}$, and the entry $(e, S) = 1$ if $e \in S$, 0 otherwise. The entries of the matrix represent the payment of the column player to the row player.

The game $G(I)$ is equivalent to the instance I of the design problem in the following way: A mixed strategy $\mathbf{x}$ of the row player in $G(I)$ corresponds to a weight distribution $\mathbf{w}$ in I. Given a mixed strategy $\mathbf{x}$ of the row player, the payment to the row player for the pure strategy S of the column player is precisely the value of the objective function $f_S(\mathbf{w})$ in I. Thus, given the row's mixed strategy $\mathbf{x}$, if the column player plays its best response to $\mathbf{x}$, then the payment to the row player is precisely $OPT_\Pi(E, \mathcal{S}, \mathbf{x})$. Finally, this means that the set of maxmin strategies of the row player in $G(I)$ is precisely the set of solutions to the instance I of the design problem $D(\Pi)$. The value of the game $G(I)$ is precisely $OPT_{D(\Pi)}(I)$.

The technique of multiplicative updates can be used to find approximate maxmin strategies of a zero-sum game much faster than by solving a linear program [FS99]. In this section we describe this technique in terms of solving instances of design problems. The algorithms and proofs here follow the proofs of [FS99] as applied to our setting. The multiplicative updates technique has proved to be extremely useful in a wide array of applications in computer science - see e.g., the recent survey paper by Arora et al. [AHK06]. We show here how this technique can be used to transform an α-approximation algorithm for an optimization problem to an α-approximation algorithm for its design version (for every α).

Algorithm Design-Linear: Given an instance $I = (E, \mathcal{S}, B)$ of a linear design problem $D(\Pi)$, the goal is to find an α-approximate solution to I. The algorithm assumes oracle access to an α-approximation algorithm $\mathcal{A}$ for the optimization problem Π.

- **Input:** Instance $I = (E, \mathcal{S}, B)$.
- **Parameters:** Real $\beta > 1$, integer $T > 1$, to be fixed later.
- **Output:** Weight vector $\overline{\mathbf{w}}$, an α-approximate solution to I.

- **Initialize** $\forall\ e : z_1(e) = 1$. Let $\mathbf{w}_1(e) = z_1(e)/\sum_e z_1(e)$.
- **Multiplicative update:** For $t = 1, \ldots, T$, do:
  - Suppose $\mathcal{A}$ on input $\mathbf{w}_t$ returns solution S_t.
  - $z_{t+1}(e) = z_t(e)\beta^{\mathbf{1}_{(e,S_t)}}$, where $\mathbf{1}_{(e,S_t)} = 1$ if S_t contains e, 0 otherwise;
  - $\mathbf{w}_{t+1}(e) = z_{t+1}(e)/\sum_e z_{t+1}(e)$
- Return $\overline{\mathbf{w}} := \frac{B}{T}\sum_{t=1}^{T}\mathbf{w}_t$

Intuitively, at each step t, the algorithm finds the minimum solution with respect to weights $\mathbf{w}_t$, and then in the next step increases the weights on the elements in the solution returned. To analyze the algorithm, following [FS99] we define the quantity *regret* as

$$R_T := \max_{\mathbf{w}:\sum_e \mathbf{w}(e)=1} \sum_{t=1}^{T} f_{S_t}(\mathbf{w}) - \sum_{t=1}^{T} f_{S_t}(\mathbf{w}_t)$$

The following theorem was proved in [FS99].

Theorem [FS99]: Fixing the choice of $\beta = 1 + \sqrt{\frac{2\ln n}{T}}$, gives us

$$R_T \leq \sqrt{T}O(\sqrt{\ln n})$$

Now we are ready to prove the bound on the quality of our solution.

Theorem 4.1

For every $\epsilon > 0$, given an α-approximation algorithm $\mathcal{A}$ to a linear minimization problem Π, algorithm Design-Linear, when run for $T = O(\frac{\ln n}{\epsilon^2\alpha^2})$ rounds, returns an α-approximate solution to every instance I of the maxmin problem $D(\Pi)$, up to an additive error $\epsilon > 0$.

Proof. We need to argue about the quantity $\min_S f_S(\overline{\mathbf{w}})$. In the following, when we use subscript $\mathbf{w}$ we assume that sum of weights is equal to 1, and that the weights will be scaled to sum to B at the end.. We follow the proof as in [FS99]. We have

$$\begin{aligned}
\min_S f_S(\overline{\mathbf{w}}) &= \min_S \tfrac{1}{T}\sum_{t=1}^{T} f_S(\mathbf{w}_t) && \text{(by linearity of } f_S\text{)}\\
&\geq \tfrac{1}{T}\sum_{t=1}^{T} \min_S f_S(\mathbf{w}_t)\\
&\geq \tfrac{1}{T}\sum_{t=1}^{T} \tfrac{1}{\alpha} f_{S_t}(\mathbf{w}_t) && (\mathcal{A} \text{ is an } \alpha\text{-approximation algorithm})\\
&\geq \tfrac{1}{\alpha}\max_{\mathbf{w}} \tfrac{1}{T}\sum_{t=1}^{T} f_{S_t}(\mathbf{w}) - O(\tfrac{1}{\alpha}\sqrt{\tfrac{\ln n}{T}}) && \text{(by Theorem of Freund-Schapire)}\\
&\geq \tfrac{1}{\alpha}\max_{\mathbf{w}}\min_S f_S(\mathbf{w}) - O(\tfrac{1}{\alpha}\sqrt{\tfrac{\ln n}{T}}) && \text{(minimum is smaller than the average)}
\end{aligned}$$

Since we finally scale the weights to sum up to B, we see that it is sufficient to run for $T = O(\frac{B^2\ln n}{\epsilon^2\alpha^2})$ rounds to get an ϵ additive error. □

Corollary 4.2

If the α-approximation algorithm $\mathcal{A}$ for Π runs in time $T_{\mathcal{A}}$, then Algorithm Design-Linear is α-approximate with additive error $\epsilon > 0$ and runs in time $O(T_{\mathcal{A}}\frac{B^2\ln n}{\epsilon^2\alpha^2})$.

4.2. Extending the framework to concave utility functions and convex cost functions. In this section, we extend the technique described in Section 4.1 to solve the design versions of convex maximization and concave minimization problems. Suppose we have oracle access to an α-approximation algorithm $\mathcal{A}$ for the optimization problem Π. We adapt the technique of gradient descent for online regret minimization introduced by Zinkevich [Zin03] and extended to the bandit setting by Flaxman et.al. [FKM05], to obtain an α-approximation algorithm for the design problem $D(\Pi)$ (with an additional arbitrarily small additive error).

Suppose the instance of the design problem is $(E, \mathcal{S}, B)$. We assume that the budget and the weights are scaled down to get $B = 1$ for notational convenience (the running time of the algorithm will depend polynomially on B). Let $n = |E|$ and let Δ denote the $n-1$ dimensional simplex, the set of all feasible weight vectors $\sum_{e \in E} \mathbf{w}_e = 1$. We assume that for all $S \in \mathcal{S}, \mathbf{w} \in \Delta$, the value of the functions $f_S(\mathbf{w})$ is bounded by a polynomial $\phi(n)$.

Algorithm Design-General:

- **Input:** Instance $I = (E, \mathcal{S}, B{=}1)$, $n = |E|$.
- **Parameters:** η, δ, ν, T to be fixed later.
- **Output:** Weight vector $\overline{\mathbf{w}}$, an approximate solution to I.
- Set $\mathbf{y}_1 = \frac{1}{n}\mathbf{1}$
- For time $t = 1, \ldots, T$, do
  - Pick a random unit vector $\mathbf{u}_t \in \mathbf{R}^n$.
    Let $\mathbf{v}_t$ be the unit vector in the direction $\mathbf{u}_t - (\mathbf{u}_t \cdot \mathbf{1})\mathbf{1}$.
  - $\mathbf{w}_t := \mathbf{y}_t + \delta \mathbf{v}_t$.
  - Run algorithm $\mathcal{A}$ with weight vector $\mathbf{w}_t$ and let it return solution S_t.
  - $\mathbf{z}_{t+1} := \mathbf{y}_t - \nu f_{S_t}(\mathbf{w}_t)\mathbf{v}_t$;
    Let $\mathbf{y}_{t+1}$ be the vector in $(1-\eta)\Delta$ which is closest to $\mathbf{z}_{t+1}$.
- Output $\overline{\mathbf{w}} := \frac{1}{T}\sum_{t=1}^{T} \mathbf{w}_t$

Following [FKM05], [Zin03] we define the regret in this setting as

$$R_T := \max_{\mathbf{w} \in \Delta} \sum_{t=1}^{T} f_{S_t}(\mathbf{w}) - \mathbf{E}[\sum_{t=1}^{T} f_{S_t}(\mathbf{w}_t)]$$

where the expectation is over the random choices of the unit vectors. Flaxman et.al. proved the following theorem

Theorem[FKM05]: For sufficiently large n and a setting of parameters of η, δ, ν,

$$R_T \leq O\left(n\phi(n)T^{5/6}\right) \tag{4.1}$$

Now we are ready to prove our bounds on the solution obtained by Algorithm Design-General. The proof is similar to the proof of Theorem 4.1 in Section 4.

Theorem 4.3

Given an α-approximation algorithm for the concave minimization problem Π, Algorithm Design-General is a randomized α-approximation algorithm, in expectation, for the design version $D(\Pi)$, with an additional arbitrarily small additive error.

Proof. Note that the weight vector $\overline{\mathbf{w}}$ returned by the algorithm is a random variable. We show that the expected cost of the minimum solution for $\overline{\mathbf{w}}$ is within α of the maxmin solution (up to additive

error). Thus we need to argue about the quantity $\mathbf{E}[\min_{S\in\mathcal{S}} f_S(\overline{\mathbf{w}})]$.

$$\begin{aligned}
\mathbf{E}[\min_S f_S(\overline{\mathbf{w}})] &\geq \mathbf{E}[\min_S \tfrac{1}{T}\sum_{t=1}^{T} f_S(\mathbf{w}_t)] && \text{(by concavity of } f_S\text{)}\\
&\geq \tfrac{1}{T}\mathbf{E}[\sum_{t=1}^{T} \min_S f_S(\mathbf{w}_t)]\\
&\geq \tfrac{1}{T}\mathbf{E}[\sum_{t=1}^{T} \tfrac{1}{\alpha} f_{S_t}(\mathbf{w}_t)] && \text{(Definition of } S_t\text{)}\\
&\geq \tfrac{1}{\alpha}\tfrac{1}{T}\max_{\mathbf{w}} \sum_{t=1}^{T} f_{S_t}(\mathbf{w}) - O(\tfrac{1}{\alpha}\tfrac{n\phi(n)}{T^{1/6}}) && \text{(by Equation (4.1))}\\
&\geq \tfrac{1}{\alpha}\max_{\mathbf{w}}\min_S f_S(\mathbf{w}) - O(\tfrac{1}{\alpha}\tfrac{n\phi(n)}{T^{1/6}}) && \text{(minimum is less than the average)}
\end{aligned}$$

Hence we see that $\overline{\mathbf{w}}$ is an α approximate (in expectation) maxmin weight distribution, with an additive error which goes to 0 as the number of rounds T becomes large compared to $n\phi(n)$ (say $T = (n\phi(n))^7$). □

4.2.1. Example: Designing graphs to minimize commute time and cover time. As an application of the framework for convex functions, we show how to design the transition probabilities on a graph to minimize the maximum commute time and cover time for a random walk on a graph. Given edge weights, a step of the standard random walk from a vertex u goes to a vertex v with probability $p(u,v) = \frac{w_{(u,v)}}{\sum_{e \text{ adjacent to } u} w_e}$. The *commute time* between u and v is defined as the expected time required to start at a vertex u and hit v and come back to u. The cover time is the expected time taken for the random walk to hit every vertex of the graph.

Here an instance $I = (E, \mathcal{S}, \mathbf{w})$ of the maximization problem Π is a graph G, with E being the edges of the graph, $\mathbf{w}$ being weights on the edges, and $\mathcal{S}$ being the collection of vertex pairs. The weights determine the transition probabilities of a random walk. The design version $D(\Pi)$ is that of assigning weights to the edges so as to minimize the maximum pairwise commute time[3].

We note that the commute time can be found in polynomial time and moreover, both are bounded by a polynomial in the number of vertices(see for example Section 6.3 [MR95]). It is also known that the commute time is a convex function of the edge weights (see [EKPS04], or Ghosh et.al [GBS05]). Thus we can solve the design problem using Algorithm Design-General. Moreover, the Matthews bound[Mat88] states that the cover time of the random walk is within a $\log n$-factor of the maximum commute time. Thus we have:

Theorem 4.4

For every graph we can find in polynomial time a weight distribution on the edges of the graph to minimize the maximum pairwise commute time of the resulting random walk. Moreover, the same distribution is a $O(\log n)$ approximation to design edge probabilities to minimize the cover time.

5. The complexity of design problems. In this section we study the relationship of the complexity of design problems and the complexity of the corresponding optimization problems.

The main result of this paper as described in Sections 3 and 4 is that solving a design problem $D(\Pi)$ is as easy as solving the corresponding optimization problem Π, for the class of concave (convex) minimization (maximization) problems, up to arbitrarily small additive errors. This is proved via two different general techniques to give Theorem 3.2 and Theorem 4.3. For linear optimization problems, if Π is in $\mathbf{P}$ then $D(\Pi)$ is also in $\mathbf{P}$ (see Remark 3.1 in Section 3.1). This may not be true for convex or concave optimization problems, since it may be that Π is in $\mathbf{P}$, but all optimal solutions for $D(\Pi)$ have

[3]In a recent result, Boyd et.al [BDX04] investigate a similar problem of assigning transition probabilities to the edges of a path such that the *mixing time* is minimized.

irrational values. However, we can still solve $D(\Pi)$ upto an arbitrarily small additive approximation in polynomial time.

A natural question to ask is if the converse also holds, i.e. whether solving the optimization problem is as easy as the design version of the same. The following simple example shows that this is not the case:

Theorem 5.1
There exists a linear minimization problem Π *such that finding the value of the minimum is* **NP***-hard, but its design version* $D(\Pi)$ *can be solved in polynomial time.*

Proof. Call a graph a bridged clique if it consists of two cliques K_1 and K_2, and two edges $(u, u'), (v, v')$ with $u, v \in K_1$ and $u', v' \in K_2$. Consider the problem of finding (the value of) the cheapest tour on a weighted bridged clique. This problem is **NP**-hard as it involves finding the cheapest hamiltonian paths between u, v and u', v' respectively. Now consider the design version of the problem. We have to find a distribution of the weight budget on a bridged clique so that the cost of the minimum weight tour is maximized. Since any tour will have to pick both edges of the bridge, the optimal strategy is to divide the weights only on the bridge edges. Thus the design version of this problem can be solved trivially in polynomial time. This construction extends to any **NP**-hard problem. □

We have seen that all design problems are as easy as their optimization versions (up to additive errors), and that some are polynomial time solvable even though the optimization versions are **NP**-hard. To complete the picture we show below that not all design problems are easy:

Theorem 5.2
There exists an **NP***-hard linear minimization problem such that the corresponding design problem is also* **NP***-hard.*

Proof. Consider the problem of finding the minimum weight Steiner tree in a weighted graph. We prove in Section 6 (Theorem 6.1) that the value of the maxmin Steiner tree is exactly the reciprocal of the maximum number of Steiner trees that can be fractionally packed in the weighted graph. However, the fractional packing number of Steiner trees is known to be **NP**-hard, as proved by Jain et al. [JMS03]. □

6. Maxmin design problems and packing problems. In this section we first show the equivalence of maxmin design problems and fractional packing problems (Theorem 6.1). Subsequently, we focus on a particular minimization problem, the Steiner tree problem. We show a systematic procedure to obtain results about fractional packing Steiner trees using the above equivalence and different existing LP relaxations for the *minimum* Steiner tree problem. Our technique is a general technique and we use the Steiner tree problem as an example since it is a well studied problem with various LP relaxations.

We start with the equivalence mentioned above. Consider a general set system, $\mathcal{F} = (E, \mathcal{S})$. The fractional packing number $\nu_f(\mathcal{F})$ is defined as the maximum number of fractionally disjoints sets in $\mathcal{S}$, that is,

$$\nu_f(\mathcal{F}) := \max\{\sum_{S\in\mathcal{S}} \lambda_S : \sum_{S:e\in S} \lambda_S \leq 1,\ \forall e \in E; \quad \lambda_S \geq 0,\ \forall S \in \mathcal{S}\}$$

Theorem 6.1
For any set system $\mathcal{F} = (E, \mathcal{S})$*, we have* $\nu_f(\mathcal{F}) = 1/OPT_{D(\Pi)}(E, \mathcal{S}, 1)$*, that is, the fractional packing number equals the reciprocal of the maxmin design of the linear instance* $(E, \mathcal{S})$ *given budget of 1.*

Proof. By duality, we can write $\nu_f(\mathcal{F})$ as

$$\nu_f(\mathcal{F}) = \min\{\sum_{e\in E} x_e : \sum_{e\in S} x_e \geq 1,\ \forall S \in \mathcal{S}; \quad x_e \geq 0,\ \forall e \in E\} \tag{6.1}$$

By definition (LP(3.1)),

$$OPT_{D(\Pi)}(E, \mathcal{S}, 1) = \max\{\lambda : \sum_{e\in S} x_e \geq \lambda,\ \forall S \in \mathcal{S}; \quad \sum_{e\in E} x_e = 1; \quad x_e \geq 0,\ \forall e \in E\} \tag{6.2}$$

We complete the proof by showing that the optimal solution of (6.1) is the reciprocal of optimal solution of (6.2). Take an optimal solution $\{x_e\}_{e\in E}$ to (6.1) of value ν_f. Note that $(1/\nu_f, \{w_e = x_e/\nu_f\}_{e\in E})$ is a feasible solution for (6.2). This is because $\sum_{e\in E} w_e = \sum_{e\in E} x_e/\nu_f = 1$ and for all sets S, $\sum_{e\in S} w_e = \sum_{e\in S} x_e/\nu_f \geq 1/\nu_f$. Similarly, if $(\lambda, \{w_e\}_{e\in E})$ is a solution to (6.2), then $\{x_e = \frac{w_e}{\lambda}\}_{e\in E}$ is a solution to (6.1) of value $\frac{1}{\lambda}$. □

6.1. Fractionally packing Steiner trees. In this section we focus on the packing problem of Steiner trees in undirected graphs. We start with some preliminaries about Steiner trees.

Preliminaries: We work with a multigraph $G = (V, E)$. The vertex set is partitioned into two sets $V = R \cup S$. R is the set of required vertices, S the set of Steiner vertices. A Steiner tree is a minimal connected subgraph of G connecting all the vertices in R. We denote the set of Steiner trees of G as $\mathcal{T}(G)$, or simply $\mathcal{T}$ when the graph is clear from context. The *integral packing number* of G is the maximum number of edge-disjoint Steiner trees in $\mathcal{T}$ and is denoted by $\nu_{int}(G)$. The *fractional packing number* of G is the maximum number of Steiner trees in $\mathcal{T}$ that can be packed fractionally and is denoted by $\nu_f(G)$. Thus,

$$\nu_{int}(G) := \max\{\sum_{T\in\mathcal{T}} \lambda(T) : \sum_{T:e\in T} \lambda(T) \leq 1, \forall e \in E; \quad \lambda(T) \in \{0,1\}, \forall T \in \mathcal{T}\} \tag{6.3}$$

$$\nu_f(G) := \max\{\sum_{T\in\mathcal{T}} \lambda(T) : \sum_{T:e\in T} \lambda(T) \leq 1, \forall e \in E; \quad \lambda(T) \geq 0, \forall T \in \mathcal{T}\} \tag{6.4}$$

Let Π, the Steiner tree polytope, be the convex hull of indicator vectors of the Steiner trees of G, that is, $\Pi := \texttt{conv}\{x \in \mathbb{R}^E : x = \mathbf{1}_T, T \in \mathcal{T}\}$. $\mathbf{1}_T$ is the vector in $\mathbb{R}^E$ with a 1 corresponding to edges in T and 0 otherwise. Given weights $w(e)$ on each edge of G, the minimum weight Steiner tree would be denoted as $MST(w)$. Note that $MST(w) = \min\{\sum_e w(e) \cdot x(e) : x \in \Pi\}$. Determining $MST(w)$ in general graphs is NP-hard. $\Pi' \subseteq \mathbb{R}^E$ is a relaxation if $\Pi \subseteq \Pi'$. The LP $\min\{\sum_e w(e) \cdot x(e) : x \in \Pi'\}$ is called an LP relaxation for the Steiner tree problem and is denoted as $L_{\Pi'}(w)$. The integrality gap, $\alpha(\Pi')$, of the relaxation is defined as the $\max_w \frac{MST(w)}{L_{\Pi'}(w)}$.

A partition $\mathcal{P} = (V_1, V_2, \cdots, V_p)$ of the vertices V is called a *valid partition* if $p > 1$ and for all $i = 1 \cdots p$, $V_i \cap R \neq \varnothing$. The edges with each end point in separate partitions are called cross-edges of the partition, $E(\mathcal{P})$. The strength of a partition is the following ratio: $\frac{|E(\mathcal{P})|}{p-1}$, The Steiner strength of the graph, denoted by $\gamma(G)$ is the minimum strength over all valid partitions.

$$\gamma(G) := \min_{valid\mathcal{P}} \frac{|E(\mathcal{P})|}{|\mathcal{P}| - 1} \tag{6.5}$$

Recall, the *max-min Steiner tree* problem is to find a weight assignment $w : E \to \mathbb{R}^+$, so that $|w| = 1$, where $|w| = \sum_{e\in E} w(e)$, and the weight of the minimum weight Steiner tree is maximized. The weight of the max-min Steiner tree is denoted as $MMST$. Thus,

$$MMST := \max_{w\in\mathbb{R}^+} \{MST(w) : |w| = 1\} = \max_{w\in\mathbb{R}^+} \{\nu : w(T) \geq \nu, \ \forall T \in \mathcal{T}; \ |w| = 1\}$$

The following observation is a special case of Theorem 6.1.

Theorem 6.2

For any graph G, $MMST(G) = \frac{1}{\nu_f(G)}$

Note that the max-min Steiner tree can be thought as

$$MMST = \max_{w:|w|=1} \min_{x\in\Pi} w \cdot x$$

To convert this into a pure maximization linear program, we take the dual of the program $L_\Pi(w) := \min_{x\in\Pi} w \cdot x$. Denote the dual as $D_\Pi(w)$. Thus, the following is an exact linear program for MMST

$$MMST := \max_{w:|w|=1} D_\Pi(w)$$

Obviously one doesn't expect an explicit characterization of $D_\Pi(w)$ given the NP-hardness of finding $MST(w)$. The observation now is that *every* LP relaxation for the minimum Steiner tree problem gives an LP relaxation for the max-min Steiner tree problem.

Let Π' be a relaxation of the Steiner tree polytope. Moreover suppose $\Pi' := \{x : Ax \geq \mathbf{1}; x \geq 0\}$ can be explicitly characterized[4]. $L_{\Pi'}(w) = \min\{w \cdot x : Ax \geq \mathbf{1}; x \geq 0\}$ is a lower bound on $MST(w)$. Taking the dual of $L_{\Pi'}(w)$, we get $D_{\Pi'}(w) = \max\{y \cdot \mathbf{1} : y^T A \leq w; y \geq 0\}$ is also a lower bound on $MST(w)$. This gives the following lower bound on $MMST$:

$$MMST \geq \max\{y \cdot \mathbf{1} : y^T A - w \leq 0; \quad |w| = 1; \quad y, w \geq 0\} \tag{6.6}$$

Moreover, if the integrality gap of the relaxation Π' is $\alpha(\Pi')$, that is, for any weight w, we have $L_{\Pi'}(w) \leq MST(w) \leq \alpha(\Pi') \cdot L_{\Pi'}(w)$, we get the integrality gap of the above relaxation is at most $\alpha(\Pi')$. This gives us:

$$MMST \leq \alpha(\Pi') \cdot \max\{y \cdot \mathbf{1} : y^T A - w \leq 0; \quad |w| = 1; \quad y, w \geq 0\} \tag{6.7}$$

Taking duals of the above LP, we get

$$MMST \leq \alpha(\Pi') \cdot \min\{\mu : Ax \geq 1; \ \mu \cdot \mathbf{1} - x \geq 0; \quad x \geq 0\} \tag{6.8}$$

In the remainder of the section we will plug in three different LP relaxations for the minimum Steiner tree problem and use the above framework and Theorem 6.2 to obtain three independently known results about the fractional packing number.

The Undirected Cut Relaxation

Let $\mathcal{U} := \{U \subsetneq V : U \cap R \neq \varnothing \text{ and } U^c \cap R \neq \varnothing\}$ denote the subsets of V which contain at least one required vertex but not all. Given a subset U of vertices, let $\delta(U)$ denote the set of edges with exactly one endpoint in U. The undirected cut relaxation polytope Π_{UC} is the following.

$$\Pi_{UC} := \{x \in \mathbb{R}^E : x(\delta(U)) \geq 1, \ \forall U \in \mathcal{U}; \ x \geq 0\}$$

Plugging this in the inequality(6.8), we get

$$MMST \leq \ \alpha(\Pi_{UC}) \cdot \min\{\mu : x(\delta(U)) \geq 1, \forall U \in \mathcal{U}; \quad \mu \geq x(e), \forall e \in E; \ x \geq 0\}$$

This implies the following theorem.

Theorem 6.3

If there are $2k$ edge-disjoint paths between any two required vertices, then $\nu_f(G) \geq k$.

Proof. By Theorem (6.2), we need to prove $MMST \leq \frac{1}{k}$. Since it is known $\alpha(\Pi_{UC}) \leq 2$, it is enough to show a feasible solution of $\frac{1}{2k}$ for the above LP. Since between any two required vertices there are $2k$ edge-disjoint paths, by Menger's theorem $|\delta(U)| \geq 2k$ for all $U \in \mathcal{U}$ and thus $\mu = x(e) = \frac{1}{2k}$ for all edges e is a feasible solution. □ This theorem was implicit in the work of Li and Li[LL03] who prove it using Mader's splitting theorem. The only tool we use is LP-duality.

Multiway Cut Relaxation

The multiway-cut relaxation polytope Π_{MC} is the following.

$$\Pi_{UC} := \{x \in \mathbb{R}^E : x(E(\mathcal{P})) \geq |\mathcal{P}| - 1, \ \forall \text{ valid } \mathcal{P}; \ x \geq 0\}$$

Plugging this in the inequality(6.8), we get

$$MMST \leq \ \alpha(\Pi_{MC}) \cdot \min\{\mu : x(E(\mathcal{P})) \geq |\mathcal{P}| - 1, \forall \text{valid } \mathcal{P}; \quad \mu \geq x(e), \forall e \in E; \ x \geq 0\}$$

This leads us to the relation between fractional packing number and the Steiner strength of the graph.

[4]More generally, Π' could be $\{x : [A \mid B](x \ t)^T \geq \mathbf{1}; x, t \geq 0\}$ where t are auxiliary variables. We omit this notation for the time being for brevity, although we will use auxiliary variables in the bi-directed cut relaxation below.

Theorem 6.4
For any graph, $\nu_f(G) \leq \gamma(G) \leq \alpha(\Pi_{MC}) \cdot \nu_f(G)$

Proof. The first inequality follows from the definition of $\nu_f(G)$ and $\gamma(G)$. The second inequality follows from the solution $\mu = x(e) = \frac{1}{\gamma(G)}$ for all e, for the above LP. The feasibility of the above solution follows from definition of $\gamma(G)$. □

Corollary 6.5
If the graph G has no Steiner vertices, that is, the Steiner trees are spanning trees, then $\nu_f(G) = \gamma(G)$.

Proof. This follows from the fact that $\alpha(\Pi_{MC}) = 1$ for graphs with no Steiner vertices [Cho89, Ful71].
□

Bi-directed cut relaxation
Call an arbitrary vertex $r \in R$ as the root. Bi-direct every edge of the multigraph and call the resulting set of arcs A giving each the same weight as the undirected edge. Let $\mathcal{U} := \{U \subsetneq V : U \cap R \neq \varnothing \text{ and } r \notin U\}$ denote the subsets of V which contain at least one required vertex but not the root. Given a set of vertices U, let $\delta^+(U)$ denote the set of arcs with heads in U and tails outside U. Consider the following polytope.

$$\Pi_{BC} := \{x \in \mathbb{R}^A : x(\delta^+(U)) \geq 1,\ \forall U \in \mathcal{U};\ x \geq 0\}$$

The bi-directed polytope is obtained by projecting onto $\mathbb{R}^E$ where the coordinate of an edge is the addition of the values on its two bi-directed arcs. We abuse notation and call the earlier polytope Π_{BC}. Note that, as was mentioned in a footnote before, Π_{BC} contains auxiliary variables: the arc variables. Thus, Π_{BC} cannot be plugged in directly into inequality(6.8), but rather into a more general inequality which can be derived similarly with the auxiliary variables. We omit the exposition of the calculation. This gives

$$MMST \leq\ \alpha(\Pi_{BC}) \cdot \min\{\mu : x(\delta^+(U)) \geq 1,\ \forall U \in \mathcal{U};\ \ \mu \geq x(e_1) + x(e_2), \forall e \in E;\ x \geq 0\} \tag{6.9}$$

where e_1 and e_2 are the bi-directed arcs of the undirected edge e.

We now digress a little to understand the forthcoming result. Consider the undirected multi-graph G as a communication network and root r as a source wishing to transmit to all the other nodes in R. Normally, such a communication is established via sending information on *multi-cast trees* which are nothing but Steiner trees. Edges have capacities (which can be modeled via multiplicity) and thus the maximum transmission rate is precisely the fractional packing number of the graph: Send $\lambda(T)$ amount of information across multicast tree T, where $\lambda(T)$ is the solution in the definition (6.3).

Using network coding, however, this throughput can be increased. The network coding principle [ACLY00] is the following: the maximum throughput achievable in a directed communication network between one source and many sinks is the minimum over all required vertices, the throughput between the source and that required vertex individually. Li and Li [LL03] noted that in an undirected graph this amounts to finding the optimum distribution of an undirected edge's capacity onto its bi-directed arcs so as to maximize the minimum throughput between the source and any required vertex. This is given via the following linear program whose optimum was called the *network coding throughput* and denoted as $\chi(G)$ [5].

$$\chi(G) := \max\{f : x(\delta^+ U) \geq f, \forall U \in \mathcal{U};\ \ x(e_1) + x(e_2) \leq 1, \forall e \in E;\ \ x, f \geq 0\} \tag{6.10}$$

Comparing the definition of $\chi(G)$ and the inequality (6.9) obtained on plugging the bi-directed cut relaxation, and noting the LP values are reciprocals of each other, the following theorem of Agarwal and Charikar [AC04] on the network coding gain ($\frac{\chi(G)}{\nu_f(G)}$) is immediate.

Theorem 6.6
[AC04] For any undirected graph G, $\nu_f(G) \leq \chi(G) \leq \alpha(\Pi_{BC}) \cdot \nu_f(G)$.

[5] One can also think of this as a fractional version of the orientation problem in directed graphs

In fact, a careful derivation shows that the second inequality holds with equality; details can be found in the paper of Agarwal and Charikar. Our goal was to present how such results can be obtained in a systematic fashion. We end this section by noting that a much more challenging direction is to obtain a relation between the fractional packing of Steiner trees and integral packing of Steiner trees; it follows from a conjecture of Kriesell [Kri03] (if true) that this ratio is at most 2, while the current best known ratio is 24 due to Lau [Lau07].

7. Discussion: Design versions of Counting Problems. In this paper we gave a systematic way of going from an optimization problem to a design problem, and we studied their relative complexity. We leave open the issue of carrying out an analogous plan of study for counting problems, in particular, #P-complete problems.

Two rather interesting design problems that arise from counting problems are the following. Determining their complexity is by itself a challenging open problem.

1. **Network reliability:** Given probabilities of edge failures in an undirected graph, the problem of determining the probability that the graph gets disconnected is #P-complete [PB83]. An FPRAS (fully polynomial randomized approximation scheme) for this problem was given by Karger [Kar99].
   Consider the following design version of this problem. Let $G = (V, E)$ be an undirected graph. With each edge e we are specified a number p_e such that $0 < p_e < 1$. We are given a total weight of W. If weight w is placed on edge e then its failure probability becomes p_e^w. The problem is to determine the optimal way of placing weight W on the edges so that the failure probability of the resulting graph is minimized.
2. **Permanent:** We will define a design version of the problem of computing the permanent of a non-negative matrix. Our problem turns out to be a generalization of the van der Waerden Conjecture, which was settled positively by Falikman [Fal81] and Egorychev [Ego81]. This conjecture states that the matrix that has all entries $1/n$ is the doubly stochastic $n \times n$ matrix that has minimum permanent.
   Let A be an $n \times n$ 0/1 matrix whose permanent is non-zero. This is the *template matrix*. The problem is to replace the entries that are 1's in A by non-negative numbers so that the permanent of the resulting matrix is the minimum possible subject to the condition that it is doubly stochastic.

REFERENCES

[AC04] A. Agarwal and M. Charikar On the advantage of network coding for improving network throughput. *Proceedings of the IEEE Information Theory Workshop*, pp. 2410 – 2424, 2004.

[ACLY00] R. Ahlswede and N. Cai and S.-Y. R Li and R. Yeung. Network information flow. *IEEE Transactions on Information Theory*, vol. IT-46, pp. 1204–1216, 2000.

[AHK06] Sanjeev Arora, Elad Hazan and Satyen Kale. The Multiplicative Weights Update Method: a Meta Algorithm and Applications. *Manuscript*, 2006.

[BDX04] S. Boyd, P. Diaconis, and L. Xiao. The fastest mixing markov chain on a graph. *SIAM Review*, 2004.

[CMV06] D. Chakrabarty, A. Mehta and V. V. Vazirani Design is as easy as optimization. *Proceedings of ICALP*, pp. 477-488, 2006

[Cho89] S. Chopra. On the spanning tree polyhedron. *Operations Research Letters*, vol. 8, pp. 35–47, 1989.

[CV02] R. Carr and S. Vempala. Randomized metarounding. *Random Struct. Algorithms 20(3)*, pages 343–352, 2002.

[Ego81] G. P. Egorychev. The solution of van der Waerden's problem for permanents. *Adv. Math., 42*, pages 299–305, 1981.

[EKPS04] J. Elson, R. Karp, C. Papadimitriou, and S. Shenker. Global synchronization in sensornets. *LATIN*, 2004.

[Fal81] D.I. Falikman. Proof of the van der Waerden conjecture regarding the permanent of a doubly stochastic matrix. *Mat. Zametki, 29*, pages 931–938, (In Russian) 1981.

[FKM05] Abraham Flaxman, Adam Tauman Kalai, and H. Brendan McMahan. Online convex optimization in the bandit setting: gradient descent without a gradient. In *SODA*, 2005.

[FS99] Y. Freund and R. Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 1999.

[FSO99] G. Frederickson and R. Solis-Oba. Increasing the weight of minimum spanning trees. *J. Algorithms*, 1999.

[FSO97] G. Frederickson and R. Solis-Oba. Efficient Algorithms for Robustness in Matroid Optimization Proceedings of *SODA*, 659–668, 1997

[Ful59] D.R. Fulkerson. Increasing the Capacity of a Network: The Parametric Budget Problem. *Management Science, 5(4)*, pages 472–483, 1959.

[Ful71] D. R. Fulkerson. Blocking and anti-blocking pair of polyhedra. *Math. Programming*, vol. 1, pp. 168–194, 1971.

[GBS05] A. Ghosh, S. Boyd, and A. Saberi. Minimizing effective resistance of a graph. *Manuscript*, Nov. 2005.

[GLS88] M. Grötschel and L. Lovász and A. Schrijver. Geometric algorithms and combinatorial optimization. Springer-Verlag, Berlin, 1988.

[Jüt06] A. Jüttner. On budgeted optimization problems. *SIAM Journal on Discrete Mathematics, 20*, pages 880, 2006.

[JMS03] K. Jain, M. Mahdian, and M. Salavatipour. Packing steiner trees. In *SODA*, pages 266–274, 2003.

[Kar99] D. Karger. A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem. *SIAM Journal on Computing, 29*, pages 492–514, 1999.

[Kri03] M. Kriesell. Edge-disjoint trees containing some given vertices in a graph. *J. Comb. Theory, Ser. B 88(1)*, pages 53–65, 2003.

[Lau07] L-C. Lau. An approximate max-Steiner-tree-packing min-Steiner-cut theorem. *Combinatorica*, vol. 27, no. 1, pp. 71–90, 2007.

[LL03] Z. Li, Z.and B. Li. Efficient computation of maximum multicast rate. *Proceedings of* 22^{nd} *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pp. 1618–1628, 2003.

[Mad78] W. Mader. A reduction method for edge-connectivity in graphs. *Ann. Discrete Math.*, vol. 3, pp. 145–164, 1978.

[Mat88] P.C. Matthews. Covering problems for Brownian motion on spheres. *Ann. Probab., 16*, pages 189–199, 1988.

[Meg79] N. Megiddo. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research, 4(4)*, pages 414–424, 1979.

[MR95] R. Motwani and P. Raghavan. Randomized Algorithms. Cambridge University Press, 1995.

[NW61] C. St. J. A. Nash-Williams. Edge disjoint spanning trees of finite graphs. *J. Lond. Math. Soc.*, 1961.

[PB83] J.S. Provan and M.O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput., 12(4)*, pages 777–788, 1983.

[Tut61] W. T. Tutte. On the problem of decomposing a graph into n connected factors. *J. Lond. Math. Soc.*, 1961.

[Zin03] M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *ICML*, pages 928–936, 2003.