Design is as Easy as Optimization

Deeparnab Chakrabarty* Aranyak Mehta† Vijay V. Vazirani‡

Abstract

We consider the class of max-min and min-max optimization problems subject to a global budget (or weight) constraint and 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 a close relationship between design problems and packing problems; this yields relationships between fractional packing of spanning and Steiner trees in a graph, the strength of the graph, and the integrality gap of the bidirected cut relaxation for the graph.

1 Introduction

In this paper, we undertake a systematic study of max-min and min-max optimization problems subject to a global budget (or weight) 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: We are given an undirected graph $G(V, E)$ and a bound B on the total weight. The problem is to find a way to distribute 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.

Juttner [Jüt06] studied a general class of problems in which such a transformation always works; he called it *budgeted 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.

*College of Computing, Georgia Tech, Atlanta GA 30332. Email: deepc@cc.gatech.edu

†Google, Inc., Mountain View, USA. Email: aranyak@google.com

‡College of Computing, Georgia Tech, Atlanta GA 30332. Email: vazirani@cc.gatech.edu

Juttner 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 problem. W.l.o.g. 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]. Besides Juttner's work, we are not aware of any systematic study of the complexity of design problems.

In retrospect, Juttner 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 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. Observe also that the design problem arising from the sparsest cut problem is not a budgeted optimization problem, since the latter is NP-hard and does not have a linear programming formulation, assuming P $\neq$ NP.

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.

Neither of these design problems is a budgeted optimization problem. In both these problems, the underlying optimization problem has only n choose 2 different possible solutions, corresponding to the number of pairs of vertices in the graph (the problem being either finding the pair with the largest mixing time or largest commute time).

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.

- 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, we observe the close relationship between maxmin design problems and the corresponding fractional packing problem. We use this to derive relations between the fractional packing number of Steiner trees and the strength of a graph. In particular, for spanning trees we prove that the fractional packing number equals the strength. The result for spanning trees can be derived using Nash-Williams and Tutte theorems [NW61, Tut61] about packing spanning trees, and ours is an alternate proof of the fact. We also derive a relationship between the fractional packing of steiner trees in a graph, the strength of the graph, and 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* $\mathcal{I}_\Pi$, and an objective function *obj*. 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 $obj(I, S)$. We restrict this to be a function of S and $\mathbf{w}$ only, so we may write this as $f_S(\mathbf{w})$, for some function f_S. For a maximization 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:

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

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.

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

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 Min-Steiner-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 Steiner 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 Steiner 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:

$$OPT_{D(\Pi)} := \max\{\lambda :\ \lambda - f_S(\mathbf{w}) \leq 0,\ \forall S \in \mathcal{S};\ \ \sum_{e \in E} \mathbf{w}(e) \leq B;\ \ \mathbf{w}(e) \geq 0,\ \forall e \in E\} \qquad (1)$$

If $f_S()$ is concave, then the above program is also 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 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.

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 1 We note that for linear optimization problems where the function $f_S()$ is linear, the program 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:

$$OPT_\Pi \geq \min \{ \ \mathbf{w} \cdot \mathbf{x} \quad \text{s.t} \quad A\mathbf{x} \geq \mathbf{b}; \ \ \mathbf{x} \geq \mathbf{0} \ \} \tag{2}$$

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(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.

[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.

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(2).

$$\max \{ \ \mathbf{b} \cdot \mathbf{y} \quad \text{s.t} \quad \mathbf{y}^T A \leq \mathbf{w}; \ \ \mathbf{y} \geq \mathbf{0} \ \} \tag{3}$$

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 LP 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 LP 3, and add the constraint that the total weight is bounded by B. Thus we solve the following LP

$$\max \{ \ \mathbf{b} \cdot \mathbf{y} \quad \text{s.t} \quad \mathbf{y}^T A - \mathbf{w} \leq \mathbf{0}; \ \ \mathbf{w} \cdot \mathbf{1} \leq B; \ \ \mathbf{y}, \mathbf{w} \geq \mathbf{0} \ \} \tag{4}$$

Let the optimal solution to LP 4 be D^*. Let $\mathbf{w}'$ be the optimum vector returned in the solution of LP 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 LP 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 $T = O(\frac{\log n}{\epsilon^2\alpha^2})$ 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,

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

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 $\mathbf{E}[\min_{S\in\mathcal{S}} f_S(\overline{\mathbf{w}})]$.

$$\begin{aligned}
\mathbf{E}[\min_S f_S(\overline{\mathbf{w}})] &\geq \mathbf{E}[\min_S \tfrac{1}{T}\sum_{t=1}^{T} f_S(\mathbf{w}_t)] && \text{(by concavity of } f_S\text{)}\\
&\geq \tfrac{1}{T}\mathbf{E}[\sum_{t=1}^{T}\min_S f_S(\mathbf{w}_t)]\\
&\geq \tfrac{1}{T}\mathbf{E}[\sum_{t=1}^{T}\tfrac{1}{\alpha} f_{S_t}(\mathbf{w}_t)] && \text{(Definition of } S_t\text{)}\\
&\geq \tfrac{1}{\alpha}\tfrac{1}{T}\max_{\mathbf{w}}\sum_{t=1}^{T} f_{S_t}(\mathbf{w}) - O(\tfrac{1}{\alpha}\tfrac{n\phi(n)}{T^{1/6}}) && \text{(by Equation 5)}\\
&\geq \tfrac{1}{\alpha}\max_{\mathbf{w}}\min_S f_S(\mathbf{w}) - O(\tfrac{1}{\alpha}\tfrac{n\phi(n)}{T^{1/6}}) && \text{(minimum is less than the average)}
\end{aligned}$$

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 on a 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 probability of moving from a vertex u to a vertex v is $p_{uv} = \frac{w_{(uv)}}{\sum_{e\sim u} w_e}$. 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 (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:

[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.

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 also gives a random walk which has $\log n$*-approximately minimum 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 **P** then $D(\Pi)$ is also in **P** (see Remark 1 in Section 3.1). This may not be true for convex or concave optimization problems, since it may be that Π is in **P**, but all optimal solutions for $D(\Pi)$ have 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 shows that this is not the case:

Theorem 5.1
There exists an linear minimization problem Π *such that finding the value of the minimum is* **NP***-complete, 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**-complete 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***-complete linear minimization problem such that the corresponding design problem is also* **NP***-complete.*

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 show the relation between fractional packing and max-min design problems for general combinatorial problems. We use this to derive relations between the fractional packing number of Steiner trees, the strength of the graph, and the integrality gap of the bidirected cut relaxation for the graph. In particular, for spanning trees we prove that the fractional packing number equals the strength. The result for spanning trees can be derived using Nash-Williams and Tutte theorems [NW61, Tut61] on packing spanning trees, and ours is an alternate proof of the fact.

6.1 Fractional Packing and Maxmin Design

Consider the tuple $\mathcal{F} = (E, \mathcal{S})$ as a general set system. The fractional packing number $k_f(\mathcal{F})$ is defined as the maximum number of fractionally disjoints sets in $\mathcal{S}$, that is,

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

Theorem 6.1
For any set system $\mathcal{F} = (E, \mathcal{S})$, we have $k_f(\mathcal{F}) = 1/OPT_{D(\Pi)}(E, \mathcal{S}, 1)$, 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 $k_f(\mathcal{F})$ as

$$k_f(\mathcal{F}) = \min\{\sum_{e\in E} x_e : \sum_{e\in S} x_e \geq 1,\ \forall S \in \mathcal{S}; \quad x_e \geq 0,\ \forall e \in E\} \tag{6}$$

By definition (LP(1)),

$$OPT_{D(\Pi)}(E, \mathcal{S}, 1) = \max\{\lambda : \sum_{e\in S} x_e \geq \lambda,\ \forall S \in \mathcal{S}; \quad \sum_{e\in E} x_e = 1; \quad x_e \geq 0,\ \forall e \in E\} \tag{7}$$

We complete the proof by showing that the optimal solution of LP 6 is the reciprocal of optimal solution of LP 7. Take an optimal solution $\{x_e\}_{e\in E}$ to LP 6 of value k_f. Note that $(1/k_f, \{w_e = x_e/k_f\}_{e\in E})$ is a feasible solution for LP 7. This is because $\sum_{e\in E} w_e = \sum_{e\in E} x_e/k_f = 1$ and for all sets S, $\sum_{e\in S} w_e = \sum_{e\in S} x_e/k_f \geq 1/k_f$. Similarly, if $(\lambda, \{w_e\}_{e\in E})$ is a solution to LP 7, then $\{x_e = \frac{w_e}{\lambda}\}_{e\in E}$ is a solution to LP 6 of value $\frac{1}{\lambda}$. □

6.2 Packing Steiner trees fractionally

In this subsection we look at the special case of Steiner trees. Given a graph $G = (V, E)$ with a set of required nodes R and Steiner nodes $S = V \setminus R$, a Steiner tree is a subtree of G containing all the nodes of R. Let τ denote the set of all Steiner trees. Let k_f denote the fractional packing number of Steiner trees. Denote the cost of the maxmin Steiner tree as `MMST`. From Theorem 6.1 we get

Theorem 6.2
The value of the maxmin Steiner tree is the reciprocal of the fractional packing number. That is, $k_f = \frac{1}{\texttt{MMST}}$.

In this section, we use the LP framework developed in Section 3.2 to relate the fractional packing number of Steiner trees to a quantity called the strength of a graph defined below.

Definition 5 A partition $\mathcal{P} = V_1 \cup V_2 \cdots \cup V_k$ of the vertices of G is called *good* if each V_i has at least one required vertex. Define the strength of this partition $\mathcal{P}$ as $\frac{|E(\mathcal{P})|}{k-1}$, where $E(\mathcal{P})$ is the subset of edges having its endpoints in different V_i's. The strength of the graph, denoted by $\gamma(G)$, is the minimum strength over all good partitions. Thus

$$\gamma(G) := \min_{\mathcal{P}=V_1,V_2,\cdots,\ \mathcal{P}\text{ good}} \frac{E(\mathcal{P})}{|\mathcal{P}|-1}$$

The following LP(8) is called the bidirected relaxation of the minimum Steiner tree problem. Bidirect each undirected edge $e = (u, v)$ of E into $\overrightarrow{(u, v)}$ and $\overrightarrow{(v, u)}$ and associate variables $x_{(u,v)}$ and $x_{(v,u)}$ respectively. Denote the set of directed edges as $\overrightarrow{E}$. Moreover, let an arbitrary vertex of R be the root denoted as r. A subset U of vertices is called *valid* if it contains a required vertex and doesn't contain the root. Let $\mathcal{U}$ denote the set of valid subsets. For $U \subset V$, let $\delta^+(U)$ denote all the directed edges (u, v) with $u \in U, v \notin U$.

$$\min\{\sum_{e\in\overrightarrow{E}} \mathbf{w}(e)x_e : \sum_{e\in\delta^+(U)} x_e \geq 1,\ \forall U \in \mathcal{U};\quad x_e \geq 0,\ \forall e \in \overrightarrow{E}\} \tag{8}$$

Let $L(\mathbf{w})$ denote the optimum of the liner program. Let α be the integrality gap of the above LP, that is, for all $\mathbf{w}$, we have $L(\mathbf{w}) \leq MST(\mathbf{w}) \leq \alpha L(\mathbf{w})$. Evaluating α is a major open problem in approximation algorithms (see for example [Vaz01]). Currently it is known $8/7 \leq \alpha \leq 2$ and when there are no Steiner vertices, $\alpha = 1$.

Following the discussion in Section 3.2, we get the following relaxation for the max-min Steiner tree problem with budget 1 and Lemma(6.3).

$$D := \max\{\sum_{U\in\mathcal{U}} y_U : \sum_{U:e\in\delta^+(U)} y_U \leq \mathbf{w}(e),\ \forall e \in \overrightarrow{E};\quad \sum_{e\in E} \mathbf{w}(e) \leq 1;\quad y_U \geq 0,\ \forall U \in \mathcal{U}\} \tag{9}$$

Lemma 6.3
If α *is the integrality gap of the bidirected cut relaxation, then* $D \leq \texttt{MMST} \leq \alpha D$

In the rest of the section we prove the relation between D and $\gamma(G)$, the strength of the graph. In particular, we show (Lemma 6.4) that for general graphs, D is between $1/\gamma(G)$ and $2/\gamma(G)$. We believe this can be strengthened, although we are unable to do so. For the particular case when there are no Steiner vertices, we can show $D = 1/\gamma(G)$ (Lemma 6.6). These, along with Theorem6.2 will imply relations between fractional packing of trees and strength.

Lemma 6.4
$\frac{1}{\gamma(G)} \leq D \leq \frac{2}{\gamma(G)}$.

Proof: $D \geq \frac{1}{\gamma(G)}$: We show a feasible solution to LP(9) of cost $\frac{1}{\gamma(G)}$. Let $\mathcal{P} = V_1, V_2, \cdots, V_k$ be the partition of minimum strength. Without loss of generality assume $r \in V_1$. Thus the set V_1 is not valid while the rest are valid. Let $y_{V_i} = \frac{1}{E(\mathcal{P})}$ for all $i > 1$. Let $\mathbf{w}(e) = \frac{1}{E(\mathcal{P})}$ for all $e \in E(\mathcal{P})$ and zero for all the rest. Thus $\sum_{e \in E} \mathbf{w}(e) = 1$. Note that the cost of this solution is $\frac{1}{\gamma(G)}$. To see that this solution is feasible, note that all zero cost edges cut none of the V_i's since they are in the partition, and each directed edge $u \to v$ cuts only the partition V_i containing u.

$D \leq 2/\gamma(G)$: Consider the dual of LP 9.

$$D = \min\{ \quad \lambda : \sum_{e \in \delta^+(U)} x_e \geq 1, \ \forall U \in \mathcal{U}; \tag{10}$$

$$x_{(u,v)} + x_{(v,u)} \leq \lambda, \ \forall (u,v) \in E; \quad x_e \geq 0, \ \forall e \in \overrightarrow{E} \quad \}$$

We now show a feasible solution to this LP of value $2/\gamma(G)$ which would prove the lemma. Consider the following solution: $x_e = 1/\gamma(G)$ for all $e \in \overrightarrow{E}$. The value of this solution is $\frac{2}{\gamma(G)}$ since for an edge $(u,v) \in E$, $x_{(u,v)} + x_{(v,u)} = 2/\gamma(G)$. For any valid set U, we have $\sum_{e \in \delta^+(U)} (1/\gamma(G)) = |\delta(U)|/\gamma(G)$, where $\delta(U)$ is the cut size of the set U in the *undirected* graph. By definition of strength, $|\delta(U)| \geq \gamma(G)$, for any valid set U, proving the lemma. □

Along with Lemma(6.3) and Theorem(6.2), this proves the following theorem:

Theorem 6.5
Fractional Packing number of Steiner trees is within 2α of the strength, where α is the integrality gap of the bi-directed cut relaxation for Steiner trees.

Lemma 6.6
In the case when there are no Steiner vertices, the value of LP(9) is exactly the reciprocal of the strength, that is, $D = 1/\gamma(G)$.

Proof: We have to show $D \leq 1/\gamma(G)$ since the other inequality is already proved in Lemma(6.4). Let the maxmin spanning tree be denoted as `MMSpT`. Firstly, from Lemma(6.3) and the fact that $\alpha = 1$ for spanning trees, we see that $D =$ `MMSpT`. Let $\mathbf{w}'$ be the cost vector such that `MMSpT` $= MST(\mathbf{w}')$. In claim 6.7, we show that $\mathbf{w}'$ has a particular structure and use it to show that $D =$ `MMSpT` $\leq 1/\gamma(G)$.

Claim 6.7
We may assume for every edge e, $\mathbf{w}'(e) = 0$ or the same value c. Moreover, the edges with $\mathbf{w}'(e) = 0$ form induced subgraphs of G.

Let E_0 be the set of edges with $\mathbf{w}'(e) = 0$. From the claim above, $G[E_0]$ defines a partition $\mathcal{P}$ of the vertices. Note that every edge in $E(\mathcal{P})$ has $\mathbf{w}'(e) = c$. Since the minimum spanning tree must uses $|\mathcal{P}| - 1$ such edges, we get `MMSpT` $= \frac{|\mathcal{P}|-1}{E(\mathcal{P})} \leq \frac{1}{\gamma(G)}$ by definition of strength. We now prove the claim.

We use the fact that the minimum spanning tree can be found via the greedy Kruskal's algorithm. Suppose the claim is false. Let $\mathbf{w}'$ be the optimal cost vector with maximum number of edges at cost 0. Let the set of these edges be denoted as E_0. If the claim is false, then there

is a set of edges E_1 at some cost c and a set of edges E_2 whose costs are strictly greater than c, with both sets nonempty. The idea now is to transform the cost vector such that we get more edges of cost 0 and still get the same minimum spanning tree.

Let $|E_1| = m_1, |E_2| = m_2$. Let T be the minimum spanning tree obtained by running Kruskal's algorithm. Thus, we pick some edges $T_i \subseteq E_i$ from each set. Let $|T_i| = t_i$ for $i = 0, 1, 2$. Consider the cost vector with costs of E_1 dropped to zero, and the extra weight distributed uniformly over edges of E_2. Thus the new cost vector is as follows: $\mathbf{w}'_1(e) = 0$ for $e \in E_0, E_1$ and $\mathbf{w}'_1(e) = \mathbf{w}'(e) + \frac{m_1 c}{m_2}$ for $e \in E_2$. Note that the order of edges with respect to the costs is not changed and thus Kruskal's algorithm on these costs will still return the same tree. The new cost of this tree is $\texttt{MMSpT} - t_1 c + t_2 \frac{m_1 c}{m_2}$. Since $\mathbf{w}'_1$ has more zero edges, cost of this tree must have decreased. Thus we get $t_2 \frac{m_1 c}{m_2} < t_1 c$ implying $\frac{m_1}{m_2} < \frac{t_1}{t_2}$.

Now consider what happens if we increase the cost of each edge in E_1 by ϵ and decrease cost of each edge in E_2 equally so that the total cost remains same. That is, we have $\mathbf{w}'_2$ such that $\mathbf{w}'_2(e) = 0$ for $e \in E_0$, $\mathbf{c}'_2(e) = c + \epsilon$ for $e \in E_1$ and $\mathbf{w}'_2(e) = \mathbf{w}'(e) - \frac{m_1 \epsilon}{m_2}$ for $e \in E_2$. Moreover, ϵ is so chosen so that $c + \epsilon \leq \mathbf{w}'(e) - \frac{m_1 \epsilon}{m_2}$ for all $e \in E_2$. In particular

$$0 < \epsilon < \frac{m_2}{m_1 + m_2} \min_{e \in E_2} (\mathbf{w}'(e) - c)$$

This can be done since $\mathbf{w}(e)$ was *strictly* greater than c for all $e \in E_2$. Thus, once again, the minimum spanning tree returned by Kruskal's algorithm on these new costs remains the same, and the cost of this tree is $\texttt{MMSpT} + t_1 \epsilon - t_2 \frac{m_1 \epsilon}{m_2} > \texttt{MMSpT}$ since $\frac{t_1}{t_2} > \frac{m_1}{m_2}$. This contradicts the optimality of $\mathbf{w}'$, and thus the first part of the claim is proved.

Recall E_0 is the set of edges with $\mathbf{w}(e) = 0$. Suppose there was a component $G[E_0]$ which was not an induced subgraph of G. Thus there was an edge $e = (u, v)$ with $\mathbf{w}(e) = c$ in a component of $G[E_0]$. Note that no minimum spanning tree would contain this edge, since it can be replaced by a zero cost path. Thus its better to shift the edge's cost elsewhere and make its cost zero. This proves the claim and completes the proof of the lemma. □

Using the fact that $\alpha = 1$ when there are no Steiner vertices, and Lemma (6.3) and Theorem(6.2), we see that $k_f(G) = \gamma(G)$. Thus,

Theorem 6.8
The fractional packing number of spanning trees equals the strength of the graph.

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

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

[BB05] Francisco Barahona and Mourad Baiou. A linear programming approach to increasing the weight of all minimum spanning trees. *INFORMS*, 2005.

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

[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.

[FIKU05] L. Fortnow, R. Impagliazzo, V. Kabanets, and C. Umans. On the complexity of succinct zero-sum games. *IEEE Conference on Computational Complexity*, pages 323–332, 2005.

[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.

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

[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.

[Lau04] L.C. Lau. An approximate max-steiner-tree-packing min-steiner-cut theorem. In *FOCS*, pages 61–70, 2004.

[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.

[Vaz01] Vijay V. Vazirani. **Approximation Algorithms**. Springer-Verlag, 2001.

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