Design is as Easy as Optimization

Deeparnab Chakrabarty[1], Aranyak Mehta[2], and Vijay V. Vazirani[1⋆]

[1] Georgia Institute of Technology, Atlanta, USA
[2] IBM Almaden Research Center, San Jose, USA

Abstract. We identify a new genre of algorithmic problems – design problems – and study them from an algorithmic and complexity-theoretic view point. We use the learning techniques of Freund-Schapire [FS99] and its generalizations to show that for a large class of problems, the design version is as easy as the optimization version.

1 Introduction

Over the last four decades, theoreticians have identified several fundamental genres of algorithmic problems and have studied their computational complexity and the inter-relationships among them. These include decision, search, optimization, counting, enumeration, random generation, and approximate counting problems. In this paper, we define and study the complexity of design problems.

This new genre of algorithmic problems should come as no surprise. In the past, several researchers have studied natural design problems – we provide some prominent examples below. Moreover, practitioners have always been faced with such problems and have sought intelligent solutions to them. However, to the best of our knowledge, this genre has not been formally defined before and subjected to a systematic complexity-theoretic study.

Every optimization problem leads to a natural design problem. This process is formally defined in Section 2. Let us illustrate it in the context of the sparsest cut problem. 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 weight of the sparsest cut is maximized. Note that this design problem is a maxmin problem.

Three examples of natural design problems considered in the past 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. Baiou and Barahona [BB05] and Frederickson and Solis-Oba [FSO99] study a cost-based design version of maximizing the minimum weight spanning tree. A closely related problem of budgeted optimization was studied by Juttner [J03]

The main result in this paper is that for a large class of optimization problems, the design version of a problem is as easy as the optimization version. We provide several different techniques to show this:

⋆ Work supported by NSF grants 0311541, 0220343 and 0515186.

- In Section 3.1, we observe that if the objective functions in the minimization (maximization) problem Π are concave (convex), then the maxmin (minmax) design problem $D(\Pi)$ can be set up as a convex optimization problem. Moreover, Π itself appears as the separation oracle required in the ellipsoid method. Thus $D(\Pi)$ is no harder than Π in terms of complexity. Further, we show using techniques of [JMS03] that if Π has an α-factor approximation algorithm, then $D(\Pi)$ also has an α-factor approximation.
- Since the ellipsoid method takes a long time in practice, we seek more efficient methods. In Section 3.2 we observe 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 linear program. If Π has an LP-relaxation which gives a factor of α, then $D(\Pi)$ also has an LP-based solution which gives factor α. In Section 3.3 we show that if the optimization problem possesses certain structural (packing) properties, then we can use these to solve the design problem more efficiently. We give examples to illustrate these specific methods.
- In Section 4, we give what is perhaps the central algorithmic result of this paper – we provide 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 as a two player zero-sum game and show that the design problem seeks the minmax value of the game. We apply the techniques of Freund-Schapire [FS99] in the additive case and that of Zinkevich [Zin03] and Flaxman et.al [FKM05] in the convex/concave cases to solve the game. This technique also requires an (approximation) algorithm for the optimization problem. If this algorithm has a worst case factor of α, then we will be able to solve the design problem upto a factor of α with an additional additive error of an arbitrarily small ϵ.
- In Section 5 we ask how hard is the design version of a problem if the optimization version is NP-hard. We provide an example in which the design version is in P and another in which the design version is NP-hard. In Section 3.1 we have established that if the optimization version is in P then so is the design version.
- In Section 6, we observe the close relationship between maxmin design problems and fractional packing of the corresponding combinatorial structures. We use this to prove some results about fractional packings of spanning and Steiner trees.

2 Problem Definition

We present a general framework to define the design version of optimization problems. An ***optimization problem*** Π consists of a set of *valid instances* $\mathcal{I}_\Pi$. Each instance I is a triple $(E_I, \mathcal{S}_I, \mathbf{w}_I)$. E_I is a universe of *elements*, and each element $e \in E_I$ has an associated weight w_e, a rational number, giving the vector $\mathbf{w}_I$. Each instance also has a set of *feasible solutions* $\mathcal{S}_I$, where each $S \in \mathcal{S}_I$ is a subset of E. The number of feasible solutions may be exponential in $|E_I|$. For an instance $I = (E_I, \mathcal{S}_I, \mathbf{w}_I)$, and a feasible solution $S \in \mathcal{S}_I$, the

objective function value for S is given as some function of the solution and the weight vector $obj(S) = f_S(\mathbf{w}_I)$. In most optimization problems like the Travelling Salesman problem, Sparsest Cut problem, etc., the function f_S is just the sum of weights of elements in S. These class of problems are called *additive* optimization problems. A more general class of problems is the one in which the functions f_S are a convex or concave function of the weight vector. In a minimization (maximization) problem one wishes to find a feasible solution of minimum (maximum) objective function value.

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 $\mathbf{w} = (w_e)_{e \in E_J}$, which satisfies the *budget constraint* $\sum_{e \in E_J} w_e \leq B_J$. 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,

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

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

3 Solving design problems

3.1 A general technique based on the ellipsoid method

Let the design problem at hand be a maxmin design problem. The analysis for minmax design problems is similar. Let $(E, \mathcal{S}, B)$ be an instance of the design problem, and let $f_S(.)$ be the function giving the objective value for the solution $S \in \mathcal{S}$. In this section, we assume $f_S(.)$ to be a concave function in the weights[3]. Consider the following program

$$\max\{ \quad \lambda \quad \text{s.t.} \quad f_S(\mathbf{w}) \geq \lambda \ \ \forall S \in \mathcal{S}; \quad \sum_{e \in E} w_e \leq B\} \tag{1}$$

Firstly note that the feasible region in the above program is convex, and thus program 1 is a convex program. This is because if $(\lambda, \mathbf{w})$ and $(\lambda', \mathbf{w}')$ are feasible solutions, then so is their convex combination: For any $0 \leq \mu \leq 1$, $\forall\ S \in \mathcal{S}$,

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

[3] $f_S(.)$ can be a concave function of the weights of all elements in E, not just the elements in S, which is used here only for indexing. Recall that we defined the special case of additive functions to have $f_S(\mathbf{w})$ as the sum of weights of elements in S.

where the first inequality uses the fact that f_S is concave.

Therefore we can use the ellipsoid method to solve the convex program. Given a candidate point $(\lambda, \mathbf{w})$, the separation oracle needs to check whether it is feasible or return a set S as a certificate of infeasibility. Note that solving the optimization problem $(E, \mathcal{S}, \mathbf{w})$ suffices: if the minimum is greater than λ then the solution is feasible, otherwise the set with the minimum objective value is the certificate of infeasibility[4]. Thus we have the following theorem:

Theorem 1. *If we have an algorithm which solves the optimization problem Π in polynomial time, then for any $\epsilon > 0$, we can solve the corresponding design problem $D(\Pi)$ up to an additive error of ϵ in time polynomial in n and $\log \frac{1}{\epsilon}$.*

Suppose we can not solve the optimization problem exactly but only have an α-approximation for it, for some $\alpha \geq 1$. That is, 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: $f_S(\mathbf{w}) \leq \alpha \min_{T \in \mathcal{S}} f_T(\mathbf{w})$. Then we can use the methods of [JMS03] to obtain an α approximation to the convex program 1.

Theorem 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.*

Note that in both Theorems 1 and 2, if the problems are additive, then we do not need the additive error of ϵ. 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. Below, we provide two techniques which are much faster and which apply if the given problem has a special structure. In Section 4, we will provide a general method which also works much faster.

3.2 A technique based on LP-relaxation

Suppose we have a linear programming relaxation for the minimization problem Π, which yields an α-approximation algorithm, for some $\alpha \geq 1$. That is, corresponding to $(E, \mathcal{S}, \mathbf{w})$ there is a linear program:

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

with the property that the optimum value of the LP, call it L, is a lower bound on the optimum of the given instance - $\forall\ T \in \mathcal{S}$, $L \leq f_T(\mathbf{w})$. Moreover,

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

there is a guarantee that for any weight vector $\mathbf{w}$, given an optimum solution to LP 2, one can produce in polynomial time a set S such that $f_S(\mathbf{w}) \leq \alpha L$.

To solve the design problem, we look at the dual of LP 2.

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

We note that the weight vector $\mathbf{w}$ is no longer in the objective function but appears in the constraints. Parametrizing the program on $\mathbf{w}$, let the optimum 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

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

Let the optimum 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 acheiving 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 $f_S(\mathbf{w}^*) \leq \alpha D(\mathbf{w}^*) \leq \alpha D^* \leq \alpha f_T(\mathbf{w}')$. Thus we have:

Theorem 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 an LP having one constraint more than that of the LP relaxation.*

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

3.3 A technique based on integral packing

Suppose we have an instance of the additive minimization problem $(E, \mathcal{S}, \mathbf{w})$ with the following structure: There exist solutions $S_1, S_2, \cdots, S_k$ which are disjoint. In this case, we see that B/k is an upper bound on the optimum of the maxmin design problem $(E, \mathcal{S}, B)$. This is because no matter how we distribute the weight vector $\mathbf{w}$, one of the sets S_i will have $\sum_{e \in S_i} w_e \leq B/k$, since these sets are disjoint. If we can demonstrate a solution of value B/k, then this is optimal.

As an example, consider the maxmin $s - t$ cut problem. If l is the length of the shortest path from s to t, we can pack l edge disjoint $s - t$ cuts, e.g. the

level cuts of the BFS tree from s to t. By the above argument, we have an upper bound of B/l on the maxmin $s-t$ cut. Now take any shortest path and distribute the weight B equally on all the edges in the path. Since any $s-t$ cut contains at least one of these edges, we see that this solution has value B/l, hence optimal.

A second example is the design version of minimum weight spanning tree in a graph - find a weight distribution to maximize the weight of an MST. Here the upper bound comes from the Nash-Williams and Tutte Theorem on packing of edge disjoint spanning trees [NW61,Tut61] and can be achieved via giving weights to the cross edges in the optimal partition. In fact, in Section 6 we shall see a close relation between maxmin design problems and fractional packing of solutions, and how the maxmin design framework can be used to prove results about fractional packing.

4 Faster algorithms for Design Problems

In this section we provide a general method to solve design problems which works much faster than the method in Section 3.1. In Section 4.1 we solve the additive case, before solving the more general concave/convex cases in Section 4.2.

4.1 Additive Design Problems, Zero-sum Games and Multiplicative Update

In the additive case, the maxmin design problem $(E, \mathcal{S}, B = 1)$ can be formulated as a two-player zero-sum game $G(E, \mathcal{S})$: The row player (the maxminimizer) has $|E|$ rows, corresponding to the elements, and the column player has $|\mathcal{S}|$ columns, corresponding to the solutions. The $|E| \times |\mathcal{S}|$ matrix has 0 or 1 entries, with the entry $(e, S) = 1 \iff e \in S$. This is the amount that the column player pays the row player.

A probability distribution on the pure row strategies corresponds to a distribution of the weight on the elements. Now the column player's best responses (in pure strategies) correspond to sets $S \in \mathcal{S}$ with mimimum weight with respect to the given distribution of weight.

Proposition 1. *The set of optimal weight distributions for the maxmin design problem $(E, \mathcal{S}, B)$ is equal to the set of maxmin strategies for the row player in the game $G(E, \mathcal{S})$, scaled by B.*

Thus the goal of the maxmin assignment problem is precisely to find a maxmin strategy for the row player. Since $|\mathcal{S}|$ may be very large, one cannot just solve the game by traditional means, say, using linear programming. We use the technique developed in [FS99] to approximate zero-sum games to approximate design problems. The algorithms and proofs remaining section mimic [FS99] in our setting.

Assume $B = 1$ for notational ease. The algorithm proceeds in rounds. In each round we define a new weight function $\mathbf{w}_t$. We assume that we have a polynomial time oracle which given any weight vector, is guaranteed to return a

solution of cost within α times the minimum cost set. That is, at each round we get a solution S_t such that $f_{S_t}(\mathbf{w}_t) \leq \alpha \min_S f_S(\mathbf{w}_t)$. Note that in this additive case, $f_S(\mathbf{w}) = \sum_{e \in S} \mathbf{w}(e)$.
We then apply the multiplicative update rule:

- Initialize $\forall\ e:\ z_1(e) = 1$. Let $\mathbf{w}_1(e) = z_1(e)/\sum_e z_1(e)$.
- **Multiplicative update:** Suppose the oracle on input $\mathbf{w}_t$ returns solution S_t. Then the new weights are found as follows:
  $z_{t+1}(e) = z_t(e)\beta^{M(e,S_t)}$, $\mathbf{w}_{t+1}(e) = z_{t+1}(e)/\sum_e z_{t+1}(e)$
  where $M(e, S_t) = 1$ if S_t contains e, 0 otherwise.

We run this algorithm for T steps. Define the regret after T steps 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 FS: $R_T \leq \sqrt{T} O(\sqrt{\ln n})$

Run the algorithm for T rounds, and take the average of all the weight vectors over the T rounds: $\overline{\mathbf{w}} := \frac{1}{T}\sum_{t=1}^{T} \mathbf{w}_t$. We prove in the next lemma that $\overline{\mathbf{w}}$ is an α approximation with additive error to the maxmin design problem. In particular we shall show

Lemma 1. $\min_S f_S(\overline{\mathbf{w}}) \geq \frac{1}{\alpha} \max_{\mathbf{w}:\sum_e \mathbf{w}(e)=1} \min_S f_S(\mathbf{w}) - O(\frac{1}{\alpha}\sqrt{\frac{\ln n}{T}})$

Proof. We follow the proof as in [FS99]. In this when we use subscript $\mathbf{w}$ we assume that sum of weights is equal to 1 and not explicitly mention it. We have

$$\begin{aligned}
\min_S f_S(\overline{\mathbf{w}}) &= \min_S \tfrac{1}{T} \sum_{t=1}^{T} f_S(\mathbf{w}_t) && (\text{ by linearity of } f_S) \\
&\geq \tfrac{1}{T} \sum_{t=1}^{T} \min_S f_S(\mathbf{w}_t) \\
&\geq \tfrac{1}{T} \sum_{t=1}^{T} \tfrac{1}{\alpha} f_{S_t}(\mathbf{w}_t) && (\text{ oracle is } \alpha \text{ approximate}) \\
&\geq \tfrac{1}{\alpha} \max_{\mathbf{w}} \tfrac{1}{T} \sum_{t=1}^{T} f_{S_t}(\mathbf{w}) - O(\tfrac{1}{\alpha}\sqrt{\tfrac{\ln n}{T}}) && (\text{ by Theorem FS}) \\
&\geq \tfrac{1}{\alpha} \max_{\mathbf{w}} \min_S f_S(\mathbf{w}) - O(\tfrac{1}{\alpha}\sqrt{\tfrac{\ln n}{T}}) && (\text{minimum} \leq \text{average})
\end{aligned}$$

Thus if we run for $T = \frac{\ln n}{\epsilon}$ rounds, we get an ϵ additive error. Thus we get the following theorem

Theorem 4. *Given a maxmin design problem $(E, \mathcal{S}, B)$, suppose we have (as a black box) an approximation algorithm which solves the corresponding minimization problem upto a factor $\alpha \geq 1$. Then we can design an algorithm which can solve the maxmin design problem upto a factor of α.*

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 maxmin (minmax) design problems with concave utility functions (convex cost functions). We use the following online optimization setting, defined in [Zin03] and modified in [FKM05]: There is an unknown collection $\mathcal{C}$ of concave utility functions over a convex feasible region F. The optimization proceeds in rounds. In round t, the algorithm has to choose a vector $\mathbf{w}_t \in F$, and then the adversary will provide a utility function $c_t \in \mathcal{C}$. The algorithm will suffer a cost of $c_t(\mathbf{w}_t)$. Zinkevich [Zin03] considered the case when the function c_t is revealed, while Flaxman et.al [FKM05] considered the bandit setting: only the value $\mathbf{c}_t(\mathbf{w}_t)$ is revealed. The regret of the algorithm after T rounds is defined as

$$R_T := \max_{\mathbf{w} \in F} \sum_{t=1}^{T} c_t(\mathbf{w}) - \sum_{t=1}^{T} c_t(\mathbf{w}_t)$$

Flaxman et.al. [FKM05] provide an algorithm called Bandit Gradient Descent (BGD), with the following guarantee:

Theorem (Flaxman et al.) If all the functions c_t defined on a set S are bounded in an interval $[-C, C]$, the regret of the BGD algorithm is

$$R_T \leq 6nCT^{5/6} \tag{5}$$

where n is the dimension of the convex set.

For our application of this setting, we will take $\mathcal{C}$ to be the collection of functions f_S of the instance $(E, \mathcal{S}, B)$ of the design problem. Suppose there exists a polytime algorithm A which given a weight vector $\mathbf{w}$ returns a solution S such that $f_S(\mathbf{w}) \leq \alpha \min_{T \in \mathcal{S}} f_T(\mathbf{w})$. For our application, we will choose the adversary to play $c_t = f_{S_t}$ where S_t is the solution returned by the algorithm A on input $\mathbf{w}_t$. We shall also assume that all functions f_S are bounded by a polynomial $C(n)$, where n is the number of elements. The BGD algorithm also requires that the convex set S has a membership oracle. For our application, the convex set will be the n-simplex corresponding to the weight distribution over the n elements. We get the following theorem whose proof is similar to the previous subsection.

Theorem 5. *Suppose we are given an approximation algorithm for the concave minimization problem Π, then we can obtain an α-approximation algorithm for the maxmin problem $D(\Pi)$.*

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.

Suppose we are given a budget B on the total weight and we have to assign weight on each edge. These 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 goal is to place weights in such a manner so that the maximum commute time among all pairs of vertices is minimized. We note that in a related result, Boyd et.al [BDX04] investigate a similar problem of assigning transition probabilities to the edges of a graph such that the *mixing time* is minimized.

We note that the commute time can be found in polynomial time (see for example [MR95]). It is also known that the commute time is a convex function of the edge weights (see [EKPS04], also [GBS06]) [5]. Thus this problem falls in the framework and thus we can apply the BGD algorithm to obtain the minmax commute time. Moreover, the Matthews bound states that the cover time is within $\log n$ of the maximum commute time. Thus we have

Theorem 6. *Given a graph and a budget B on the total edge weights, one can find (upto additive error) a weight distribution on the edges so that the maximum commute time between two vertices is minimized over all possible weight distributions. Moreover, the same distribution also gives a $\log n$ approximation to the minimum cover time over all possible distributions.*

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 (upto arbitrarily small additive errors). This is proved via two different general techniques to give Theorem 2 and Theorem 5.

A natural question is if the converse also holds, i.e. whether the complexity of the optimization and design version of a problem are the same. The following shows that this is not the case:

Theorem 7. *There exists an additive 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.*

[5] Note the problem is a *minmax* problem and hence we require convex objective functions

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 trivally 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, and that some are polynomial time sovable even though the optimization versions are **NP**-hard. To complete the picture we show below that not all design problems are easy:

Theorem 8. *There exists an* **NP***-complete additive 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 9) 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].

We mention here a related result of Fortnow et al. [FIKU05], in which they study the complexity of solving a succinctly represented zero-sum game. Our setting is different in that the number of row strategies is part of the input size, and we have access to an (approximately) best-response oracle.

6 MaxMin Design problems and Packing problems

For lack of space we defer most of the definitions and proofs of this section to the full version of the paper, while providing a sketch of the main results.

6.1 Fractional packing and maxmin design

Consider an instance $(E, \mathcal{S}, 1)$ of a design problem with a budget of 1. A collection of sets $S_1, S_2, \cdots, S_k \in \mathcal{S}$ are said to pack fractionally with weights $\lambda_1, \cdots, \lambda_k$, if for each element e, $\sum_{S_i : e \in S_i} \lambda_{S_i} \leq 1$. The value of the packing is $\sum \lambda_i$.

Theorem 9. *Given a set of elements E and a collection of subsets $\mathcal{S}$ of E, the maximum number of sets that can be packed fractionally is exactly equal to the reciprocal of the maxmin design of the additive instance $(E, \mathcal{S}, 1)$.*

Proof. (Sketch) The LPs for fractional packing and maxmin design are duals of each other upto taking reciprocals.

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 in R. Let τ denote the set of all Steiner trees in G. Let k_f denote the maximum number of Steiner trees that can be packed fractionally. Thus

$$k_f = \max\{\sum_{T \in \tau} \lambda_T \quad \text{s.t.} \quad \forall e \in E : \sum_{T : e \in T} \lambda_T \leq 1\}$$

We shall call this the fractional packing number for Steiner trees. 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 via the well-known bidirected LP relaxation for minimum weight Steiner tree.

Given a partition P of vertices with a required vertex in each partition, the strength of a partition $\gamma(P)$ is defined as the ratio of the number of cross-edges and the size of the partition minus 1. The strength of a graph, γ is defined as the minimum over all partitions. The bidirected-cut relaxation is an LP-relaxation for the minimum Steiner tree problem (see e.g. [Vaz00]). Evaluating the integrality gap α of this relaxation is a major open problem and currently it is known that $8/7 \leq \alpha \leq 2$. We prove the following result.

Theorem 10. *Fractional Packing number of Steiner trees is within* 2α *of the strength, that is,* $\frac{\gamma}{2\alpha} \leq k_f \leq \gamma$.

The proof proceeds by proving that the *maxmin Steiner tree* is within 2α of the reciprocal of the strength and by Theorem 9 we are done. This is proved by giving feasible solutions to the LP relaxations obtained from the bidirected-cut relaxation, as in Section 3.2.

For the special case of spanning trees, we prove a stronger result.

Theorem 11. *The fractional packing number of spanning tree is exactly the strength of the graph.*

The proof uses the same techniques as the last proof and the fact that the spanning tree can be found via a greedy algorithm. All these proofs can be found in the full version of the paper and have been omitted here for sake of brevity.

Remark: We note that Jain et.al [JMS03] proved that evaluating the fractional packing number is **NP**-hard, and an α-approximation to the minimum Steiner tree problem implies existence of an α-approximation to the fractional packing problem. We note that they do not show any relation to the strength of the graph while Theorem 10 wishes to investigate the relationship with strength. Also, Theorem 11 can be directly inferred from the Nash-Williams and Tutte theorems [NW61,Tut61], but our proof techniques do not use these theorems.

We find it an interesting question as to whether the relationship between packing and maxmin design problems can be used to produce other intersting

packing theorems in other combinatorial settings. Regarding the Steiner tree setting, it follows from a conjecture of Kriesell [Kri03], that the maximum number of Steiner trees that can be packed *integrally* is within 2 times the strength of the graph. Recently, Lap Chi Lau [Lau04] has proved this within a factor of 26. Improving this factor and settling Kriesell's conjecture seems to be a challenging problem.

References

[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.

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

[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*, 29:79–103, 1999.

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

[GBS06] A. Ghosh, S. Boyd, and A. Saberi. Minimizing effective resistance of a graph. *Manuscript*, 2006.

[JÖ3] A. Jüttner. On budgeted optimization problems. *Proc. 3rd Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications*, pages 194–203, 2003.

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

[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.

[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.

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

[Vaz00] Vijay V. Vazirani. **Approximation Algorithms**. Springer, 2000.

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