Electronic Colloquium on Computational Complexity, Report No. 110 (2014)

Separation between Estimation and Approximation

Uriel Feige [*] Shlomo Jozpeh [*]

August 19, 2014

Abstract

We show (under standard assumptions) that there are NP optimization problems for which estimation is easier than approximation. Namely, one can estimate the value of the optimal solution within a ratio of ρ, but it is difficult to find a solution whose value is within ρ of optimal. As an important special case, we show that there are linear programming relaxations for which no polynomial time rounding technique matches the integrality gap of the linear program.

1 Introduction

We briefly review some basic concepts related to approximation algorithms for optimization problems. For simplicity of the presentation, we shall only deal with maximization problems, though our results can be adapted to minimization problems as well. Given a binary relation (also referred to as *predicate*) R, we say that y is a feasible solution for x if $(x, y) \in R$. Let $|s|$ denote the representation size of input s (e.g., the number of bits in some standard binary representation). We say that R is a polynomial relation if for every x, every feasible y is of size $|y| \leq |x|^{O(1)}$, and furthermore, there is a polynomial time algorithm that for every pair (x, y) decides whether $(x, y) \in R$. Let V be a value function, mapping pairs (x, y) to nonnegative integer values, and computable in polynomial time. For simplicity of the presentation and essentially without loss of generality, let us assume that $V(x, y) = 0$ if y is not feasible for x. An NP-optimization problem is specified by a predicate R and value function V as above. An instance of the problem is an input x, and the desired goal is to output a y maximizing $V(x, y)$. Given such an x, we refer to such a y as an optimal solution, and to the corresponding value of $V(x, y)$ as the optimum value $\mathrm{opt}(x)$.

For example, in the maximum independent set problem, the input x is a graph, $(x, y) \in R$ iff y is an independent set in x, and $V(x, y)$ outputs the number of vertices in y if y is a feasible solution, and outputs 0 otherwise.

[*]Weizmann Institute of Science. Email: {uriel.feige,shlomo.jozeph}@weizmann.ac.il

ISSN 1433-8092

Given that some NP-optimization problems cannot be solved optimally in polynomial time (unless P=NP), a common approach of coping with this difficulty is via an approximation algorithm. Given R and V as above, a polynomial time algorithm A is said to have *approximation ratio* $0 \leq \rho \leq 1$ if for every input x it outputs a solution $A(x)$ whose value satisfies $V(x, A(x)) \geq \rho \cdot \text{opt}(x)$. A polynomial time algorithm B is said to have *estimation ratio* $0 \leq \rho \leq 1$ if for every input x it outputs a value $B(x)$ satisfying $\rho \cdot \text{opt}(x) \leq B(x) \leq \text{opt}(x)$. Clearly, approximation is at least as difficult as estimation, because a ρ-approximation algorithm A can be used as a ρ-estimation algorithm B by taking $B(x) = V(x, A(x))$. The question addressed in this work is whether there are NP-optimization problems for which estimation is easier than approximation, in the sense that the best estimation ratio for the problem is strictly larger than the best approximation ratio. We refer to this as a separation between estimation and approximation.

Observe that if P=NP then every NP-optimization problem can be solved exactly, and there is no separation between estimation and approximation. Hence to establish a separation we need to at least assume that P does not equal NP. In fact, to establish our results, we shall use an even stronger assumption. A polynomial relation R is in TFNP [9] (Total Function NP) if every x has a feasible solution. A TFNP relation is said to be in FP if there is a polynomial time algorithm that on input x finds a feasible solution. It is currently not know whether every TFNP predicate is in FP, and there are several TFNP relations that serve as plausible candidates for not being in FP. Examples include *factoring* (every integer has a prime factorization, but no polynomial time factoring algorithm is known, a fact that serves as the basis of several well known cryptographic schemes, such as RSA), *Nash equilibrium* (every two player game in normal form has a mixed Nash equilibrium, but finding a Nash equilibrium is PPAD-complete [2]), and *maximal cut* (every edge weighted graph has a cut whose size cannot be improved by a single vertex changing size, but finding a maximal cut is PLS-complete [7]).

We observe that a separation between estimation and approximation can be established under the assumption that TFNP is not in FP. In fact, one can precisely control the gap between estimation and approximation.

Theorem 1. *For every $0 \leq \alpha < \beta \leq 1$ and $\epsilon > 0$, there is an NP-optimization problem that has a β-estimation algorithm, an α-approximation algorithm, no $(\beta + \epsilon)$-estimation algorithm (unless P=NP), and no $(\alpha + \epsilon)$-approximation algorithm (unless TFNP is in FP).*

The NP-optimization problem in the proof of Theorem 1 is somewhat artificial and was designed specifically for the proof. For natural NP-optimization problems, such a separation can often be proved not to occur. See Section 1.1 for more details.

A common technique for designing approximation algorithms is via linear programming relaxations. One first formalizes the NP-optimization problem π as an integer linear program (IP). This step is typically not difficult, because integer linear programming is NP-hard (and hence other problems in NP can be reduced to it). Thereafter, one relaxes the integrality constraints, thus achieving a linear program (LP) relaxation whose optimal value satisfies $\mathrm{opt}_{LP}(x) \geq \mathrm{opt}(x)$. The smallest possible ratio $\frac{opt(x)}{\mathrm{opt}_{LP}(x)}$ (over all x) is referred to as the integrality gap ρ_{LP} of the LP. As linear programs can be solved in polynomial time, computing $\mathrm{opt}_{LP}(x)$ (and scaling the result by ρ_{LP}) can serve as a ρ_{LP}-estimation algorithm for π. To obtain an approximation algorithm one designs a *rounding* procedure that takes a (fractional) solution of the LP and "rounds" it to an integer solution to the integer program. Let $\mathrm{rounded}(x)$ denote the value of the solution obtained after rounding the optimal LP solution. Ideally, one can establish that the value $\frac{\mathrm{rounded}(x)}{\mathrm{opt}_{LP}(x)} \geq \rho_{LP}$, which then implies that computing $\mathrm{rounded}(x)$ is a ρ_{LP}-approximation algorithm for π. This approach has been successful numerous times (see [6, 14, 15] for many examples), and hence one might be led to believe that for every LP relaxation there is a polynomial time rounding procedure that matches the integrality gap. Our next theorem specializes Theorem 1 to the context of integer programs, showing that rounding procedures that match the integrality gap do not always exist.

Theorem 2. *For every $0 \leq \alpha < \beta \leq 1$ and $\epsilon > 0$, there is a family of (maximization) integer programs with non-negative objective function with the following properties:*

1. *For every integer program its LP-relaxation has an integrality gap of no worse than β.*
2. *For infinitely many integer programs the LP-relaxation has an integrality gap no better than $\beta + \epsilon$.*
3. *There is a polynomial time algorithm that gives a solution whose value approximates the optimal integer program value within a ratio of α.*
4. *There is no $(\alpha + \epsilon)$-approximation algorithm for the optimal solution of the integer program, unless TFNP is in FP.*

The family of integer programs in Theorem 2 defines an NP-optimization problem, and hence Theorem 2 implies Theorem 1.

Our separation between estimation and approximation (Theorems 1 and 2) assumes that TFNP is not contained in FP. This assumption is unavoidable, as otherwise no separation exists.

Proposition 3. *If α-estimation of a value function V is in FP, then α-approximation of V is reducible to a problem in TFNP.*

1.1 Related Work

The current work continues a line of research outlined in [3]. The reader is referred to [3] for a more detailed discussion of the motivations and background (though note that there are more recent developments that are not covered in [3] and are mentioned below). Here we provide a shorter presentation of the background.

The distinction between estimation and approximation is analogous to the distinction between decision (does a solution exist?) and search (find a solution) for NP-problems. For NP-complete problems, due to their self reducible nature, search can be reduced to decision, and there is no separation between decision and search. However, for problems in NP that are not (known to be) NP-complete, it is plausible that such a separation exists. In particular, the class TFNP (mentioned in Section 1) contains problems that always have solutions (hence decision is trivial), but for which the search problem is not known to be solvable in polynomial time. Unlike the class NP which has NP-complete problems (such as SAT), TFNP does not seem to have complete problems (for reasons that we shall not discuss here), and hence many subclasses of TFNP has been introduced (such as PLS [7] and PPAD [11]), and these subclasses do have complete problems. In a sense, our Theorem 2 can be thought of as suggesting for TFNP a problem that can serve the purpose usually attributed to complete problems. Namely, one may think of the problem of rounding LP-relaxations within ratios that match the integrality gap as a problem that is essentially in TFNP (it is reducible to a problem in TFNP, by proposition 3), and every problem in TFNP is reducible to it.

Known NP-hardness of approximation results are in fact (we are not aware of exceptions) hardness of estimation results. Hence in those cases (which are quite common by now) in which the hardness ratio matches the approximation ratio given by known algorithms (such as max 3SAT, where this ratio is $7/8$ up to low order terms), there is no separation between estimation and approximation (except possibly in the low order terms, an issue that is beyond the scope of the current paper).

There are only a few optimization problems in which the current state of our knowledge is such that the estimation ratios known are better than the approximation ratios know. Some such problems were discussed in [3], but even for some of these problems there had since been progress in closing the apparent gap between estimation and approximation. Most notable, new algorithmic versions of the local lemma [10, 4], of discrepancy bounds [1, 8] and of local search [13] serve towards this purpose.

2 Formal Definitions and Proofs

2.1 Definitions

Definition 4. Let h be any polynomial. Let $D_h = \bigcup_n \{0,1\}^n \times \{0,1\}^{h(n)}$. A function $V : D_h \to \mathbb{N}$ that is computable in FP (Function Polynomial Time) is a *value function.*

We may define a value function on inputs satisfying a certain property with the implicit intention that the function gives a value of zero to all other inputs.

Definition 5. Given a value function V and an input x, let $M_x^V = \max_y \{V(x,y)\}$. An *$\alpha$-solution* for V and x is a string z such that $V(x,z) \geq \alpha M_x^V$

Definition 6. *α-estimation* of a value function is the following problem: given a value function V and a string x, output a value v such that $\alpha M_x^V \leq v \leq M_x^V$.

Definition 7. *α-approximation* of a value function is the following problem: given a value function V and a string x, output an α-solution.

Example 8. The value function $V_{\mathrm{SAT}}(\varphi, a)$ is defined as follows: interpret φ as an E3SAT formula, and a as its assignment. Return the number of satisfied clauses. It is known that $7/8$-approximation is in FP, while $(7/8 + \epsilon)$-estimation is NP-hard [5], for any $\epsilon > 0$.

Definition 9. A relation R is in the complexity class TFNP (Total Function NP) if there is a polynomial time Turing machine T and a polynomial h such that $R = \{(x, y) \mid |y| \leq h(|x|), T(x, y) = 1\}$ and for any string x there is a y such that $(x, y) \in R$. The first element of the pair is called the input, and the second element is called the solution. The goal of every problem in TFNP is to find a solution, given an input. Note that given any pair (x, y), it is possible to use T to verify that $(x, y) \in R$ in polynomial time.

Definition 10. A reduction from a certain relation R to a TFNP problem S is a pair of polynomial time computable functions (f, g). It must hold that for any x, if $(f(x), y) \in S$, then $(x, g(x, y)) \in R$.

Remark 11. Note that if there is a reduction from R to S then a black box solving S can solve R, but R is not necessarily in TFNP; Every input has a solution that can be verified to be in R using a specific polynomial Turing machine, but some input-solution pairs may not be verifiable to be in R.

Note that if α-approximation for some value function V is in FP, α-estimation of V is in FP as well. If α-estimation is NP-hard, then α-approximation is also NP-hard. Additionally, For any $\beta \leq \alpha$

- If α-estimation (approximation) of V is in FP, then β-estimation (approximation) of V is in FP.
- If β-estimation (approximation) of V is NP-hard, then α-estimation (approximation) of V is NP-hard

Most hardness of approximation results show that value functions are NP-hard to estimate, which also shows that value functions are NP-hard to approximate. On the other hand, usually algorithms in P approximate, and not just estimate.

2.2 Proofs

Given a value function V, suppose that α-estimation is in P. How hard can α-approximation be? We now prove Proposition 3 which addresses this question.

Proof. Let V be a value function, and let f be a function in FP that α-estimates V. Define the following relation: $R = \{(x,y) \,|V(x,y) \geq f(x)\}$. Note that for any input x, there is some solution y such that $(x,y) \in R$. Additionally, given x and y, one can verify in polynomial time whether $(x,y) \in R$ or not. Therefore, the relation R is in TFNP. α-approximation of V is reducible to finding a string in the relation R, hence, α-approximation is reducible to TFNP. □

Remark 12. Unlike the relation $R = \{(x,y) \,|V(x,y) \geq f(x)\}$, the relation $\hat{R} = \left\{(x,y) \,|V(x,y) \geq \alpha M_x^V\right\}$ might not be in TFNP (because for a given x it may include solutions y for which $\alpha M_x^V \leq V(x,y) < f(x)$ and then membership of $(x,y) \in \hat{R}$ might not be decidable in polynomial time).

The following proposition is a special case of Theorem 1 and illustrates the proof techniques that can be used in order to prove Theorem 1. Later, the proof of Theorem 2 will serve as a complete proof for Theorem 1.

Proposition 13. *For any $R \in \mathsf{TFNP}$, there is a value function such that $7/8$-estimation is in P, $(7/8 + \epsilon)$-estimation is NP-hard, $7/16$-approximation is in P, and $(7/16 + \epsilon)$-approximation is as hard as R, for any $\epsilon > 0$*

Proof. Given $R \in \mathsf{TFNP}$, we define the following value function V_R: Given (x,y), the input x is interpreted as encoding two sub-inputs $x = (\varphi, r)$, where φ is interpreted as an E3SAT formula and r as an input to R, and the solution y is interpreted as encoding two sub-solutions $y = (a,s)$, where a is an assignment for φ and s is a solution for r.

Let c_φ^a be the number of clauses in φ satisfied by a.

$$V_R(\varphi, r, a, s) = \begin{cases} c_\varphi^a & (r,s) \notin R \\ 2c_\varphi^a & (r,s) \in R \end{cases}$$

By the definition of TFNP, for any string r there is a string s such that $(r,s) \in R$. Thus, if there is an assignment satisfying c clauses in φ, the value function can get value $2c$. Since every E3SAT formula has an assignment satisfying $7/8$ of its clauses, we have that $7/8$-estimation is in FP. Since deciding whether a general 3SAT formula is $7/8 + \epsilon$ satisfiable or completely satisfiable is NP-hard [5], $(7/8 + \epsilon)$-estimation is NP-hard.

It is easy to find an assignment that satisfies $7/8$ of the clauses of any given E3SAT formula, but this gives us only $7/16$-approximation to V_R, unless we can solve R. If a black box gives us β-approximation of R, for any $\beta > 7/16$, either the black box approximated φ to within a factor better than $7/8$, or it found a solution for r. Since $(7/8 + \epsilon)$-approximation is an NP-hard problem, solving any TFNP problem (including R) can reduced to it, so β-approximation is at least as hard as R. □

We now turn to prove Theorem 2. Its proof is broken into several propositions.

Proposition 14. *For any $R \in \mathsf{TFNP}$, there is a set of integer programs (one for each input instance x) where*

- *The integer program has a feasible solution, and a feasible solution can be found in polynomial time.*
- *Every feasible solution has nonnegative value.*
- *The maximum value of the integer program is 1.*
- *The linear program relaxation (that is, replacing the requirement that the variables are in $\{0,1\}$ with the requirement that they are in the range $[0,1]$) has maximal value 1. Hence the integrality gap is 1.*
- *Outputting a solution to the IP of value that is not 0 is as hard as solving R.*

Proof. Given $R \in \mathsf{TFNP}$ let T denote the Turing machine associated with R. For every n, one can associate a circuit C_n that simulates the computation of T on those input pairs (x, y) of size $|x| = n$, and outputs 1 if and only if $(x, y) \in R$. As binary NAND forms a complete basis for logical gates, we may assume that all gates in the circuit are NAND gates. Given an input string x' with $|x'| = n$, let $C_n(x')$ denote the NAND circuit obtained from C_n by simplifying, after the input variables x are fixed to x'. Only the variables y remain as input to $C_n(x')$. We associate a variable name μ with the output of the circuit.

We now associate a 0/1 integer program IP with circuit $C_n(x')$. Every input (a y variable) to the circuit and every output of every gate will have an IP variable associated with it. The variable associated with the output gate of the circuit is called μ, and the objective of the IP is to maximize μ. The constraints are as follows. For every variable z we have the 0/1 constraint $z \in \{0, 1\}$. For every NAND gate the IP contains two constraints as follows. Let z_1 and z_2 (which need not be distinct) denote the input variables to the NAND gate, and let the output variable of the gate be z_3. The constraints associated with the gate are:

- $z_1 + z_2 + z_3 \leq 2\mu$.
- $z_1 + z_2 + 2z_3 \geq 2\mu$.

Observe that if $\mu = 0$ the assignment $z_i = 0$ for $i \in \{1, 2, 3\}$ satisfies the constraints. However, if $\mu = 1$ only assignments consistent with the NAND gate satisfy both constraints. Specifically, if $z_1 = z_2 = 1$ then the second constraint is satisfied and the first constraint requires that $z_3 = 0$. For other assignments to z_1, z_2, the first constraint is satisfied and the second constraint requires that $z_3 = 1$.

The IP is feasible, and assigning all variables to 0 is a feasible solution (of value 0). Every feasible solution has nonnegative value because of the 0/1 constraints. The maximum value of the IP is 1, obtained by considering a solution y such that the output of $C_n(x')$ is 1 (such a y necessarily exists, because $R \in \mathsf{TFNP}$), and assigning the values of variables in the IP according to the computation of $C_n(x')$ on input y. The linear programming relaxation (relaxing $z \in \{0,1\}$ to $0 \leq z \leq 1$ for all variables) has value at most 1 because of the constraint $\mu \leq 1$. Assigning $\mu = 1$ and $z = \frac{1}{2}$ to all other variables is a feasible solution of value 1 for the LP relaxation.

For every feasible solution of value $\mu = 1$ to the IP, the value of the y variables is such that $(x', y) \in R$. Hence finding a solution to the IP of value 1 is as hard as solving R. □

Proposition 15. *For any $0 < \gamma < 1$, there is a set of integer programs satisfying*

- *The integer program is feasible and the value of every feasible solution is nonnegative.*
- *The maximum value of each program is between γ and 1.*
- *A solution with value γ can be found in polynomial time.*
- *The linear program relaxation (that is, replacing the requirement that the variables are in $\{0,1\}$ with the requirement that they are in the range $[0,1]$) has maximal value exactly 1. Hence the integrality gap is no worse than γ.*
- *Estimating the optimal value of the integer programs within a ratio of $\gamma + \epsilon$ is NP-hard.*

Proof. Consider an arbitrary 3CNF formula with m clauses over a set x of n variables. Represent it as an integer program in the following way (which is standard except for our choice of objective function). For each variable x_j the IP has a corresponding variable, and for every clause j the IP has a variable y_j. All variables are in $\{0,1\}$. For every clause there is a linear constraint that allows the corresponding y variable to be 1 if and only if the clause is satisfied by the x variables. For example, if clause j is $(x_1, \bar{x}_2, x_3)$ then the corresponding constraint is $x_1 + (1 - x_2) + x_3 \geq y_j$. The IP also has an auxiliary variable μ with the constraint $\mu = 1$. The objective function is to maximize $(8\gamma - 7)\,\mu + \frac{8(1-\gamma)}{m}\Sigma_{j=1}^m y_j$, which by the constraint $\mu = 1$ is equivalent to $8\gamma - 7 + \frac{8(1-\gamma)}{m}\Sigma_{j=1}^m y_j$. To ensure nonnegativity of the objective value, we add the constraint $\Sigma_{j=1}^m y_j \geq 7m/8$.

One easily observes that to maximize the objective function one needs to maximize $\Sigma_{j=1}^m y_j$. As for every 3CNF formula with m clauses there is an assignment satisfying at least $7m/8$ clauses, the IP is feasible. Its maximum value is thus at least $8\gamma - 7 + \frac{8(1-\gamma)}{m}\frac{7m}{8} = \gamma$. If the 3CNF formula is satisfiable then the LP has a feasible solution with $\Sigma_{j=1}^m y_j = m$, and then the maximum value is $8\gamma - 7 + \frac{8(1-\gamma)}{m} m = 1$. No higher value is possible neither for the IP nor for its LP relaxation because of the constraints $y_j \leq 1$. The LP relaxation has value 1, by taking $x_i = \frac{1}{2}$ for every i, and $y_j = 1$ for every j.

By the construction of the IP, for any 0/1 solution, the x variables are an assignment satisfying Σy_j clauses of the 3CNF instance. Since it is NP-hard to distinguish between satisfiable 3CNF-formulas and those that are only $\frac{7+\epsilon}{8}$-satisfiable [5], estimating the optimal value of the integer program within a ratio of $\gamma + \epsilon$ is NP-hard. □

We remark (for the purpose of handling a technicality in the proof of Theorem 2) that Proposition 15 also holds when $\gamma = 1$, in a trivial sense. (Take the IP with one constraint $x = 1$ and objective function x. Moreover, estimating the optimal value of the integer program within a ratio of $\gamma + \epsilon = 1 + \epsilon$ is then impossible, rather than just NP-hard, as approximation ratios cannot exceed 1.)

We now prove Theorem 2.

Proof. The case $\alpha = 0$ and $\beta = 1$ is handled by Proposition 14. Hence we may assume that $1 - \beta + \alpha > 0$.

Suppose that there is $R \in \mathsf{TFNP}$ that is not in P (given x, there is no polynomial time algorithm guaranteed to find y such that $(x, y) \in R$). For such an R and an input x', consider the corresponding integer program from Proposition 14. Call it IP_1 and its objective value v_1. Given a 3CNF formula, consider the corresponding integer program from Proposition 15, with $\gamma = \frac{\alpha}{1-\beta+\alpha}$ (note that necessarily $\gamma \leq 1$). Call it IP_2 and its objective value v_2.

The integer program IP is a concatenation of the constraints of IP_1 and IP_2 on disjoint sets of variables, and its objective function is $\lambda v_1 + (1 - \lambda)v_2$, where $\lambda = \beta - \alpha$. We remark that given an IP of this form, one can easily decide which variables originated from IP_1 and which variables originated from IP_2. (For example, the constraints involving the former have a variable whose coefficient is 2.)

Proposition 14 implies that finding a feasible solution (of value at least 0) for IP_1 is easy, and Proposition 15 implies that finding a solution of value γ for IP_2 is easy. Hence finding a solution of value $(1 - \lambda)\gamma = \alpha$ for IP is easy. Finding a solution of value above $\alpha + \epsilon$ is as hard as solving R (because it requires either solving R, or approximating 3SAT within a ratio better than 7/8, which is NP-hard and hence at least as hard as solving R). There is always a solution of value 1 for IP_1 and hence a solution of value $\lambda + (1 - \lambda)\gamma = \beta$ for IP. The LP-relaxation has value 1 (because this is true for both IP_1 and IP_2), and hence the integrality gap is no worse than β. For infinitely many inputs IP_2 does not have value above $\gamma + \epsilon$ (otherwise estimating its value within this ratio could not be NP-hard), hence there are infinitely many integer programs in the IP family whose LP-relaxation has integrality gap no better than $\beta + O(\epsilon)$. $\square$

2.3 Extensions

There are known reductions between various subclasses of TFNP (see [12]). These subclasses can be incorporated into more elaborate constructions based on the proof method of Proposition 13.

Let $\mathcal{R} = \{R_i\}_{i=1}^{k}$ be relations in TFNP such that there is a reduction from R_i to R_{i+1}. The input and the solution are treated as $k+1$ strings, $x = (\varphi, r_1, \cdots, r_k)$ and $y = (a, s_1, \cdots, s_k)$. φ is interpreted as a E3SAT formula. Let c be the number of clauses in φ satisfied by a. For all $1 \leq i \leq k$, $p_i = 2$ if $(r_i, s_i) \in R_i$, otherwise, $p_i = 1$. We define the value function $V_{\mathcal{R}}(\varphi, r_1, \cdots, r_k, a, s_1, \cdots s_k) = c\Pi p_i$. Since each p_i can be made to be 2, $7/8$-estimation is in FP. However, for $0 \leq j \leq k - 1$

$\left(\frac{7}{2^{3+j}} + \epsilon\right)$-approximation is at least as as hard as R_{k-j}: Let x_{k-j} be an input for R_{k-j}. Let x_{i+k-j} the reduction of x_{k-j} to R_{i+k-j}. Let φ be an E3SAT formula that is always $7/8$-satisfiable. Suppose that given the input $(\varphi, x_1, \cdots, x_k)$ (where the strings x_t for $t < k-j$ are arbitrary), we have a $\left(\frac{7}{2^{3+j}} + \epsilon\right)$-approximation $(a, s_1, \cdots s_k)$. There is a solution of value $7 \cdot 2^{k-3}$, so the approximation has a value of at least $7^2 2^{k-j-6}$. Any solution must have the form $7 \cdot 2^p$ for some integer p, so the approximation must have value of at least $7 \cdot 2^{k-j-3}$. Hence, $k-j$ of the pairs (x_i, s_i) must be in their respective R_i, and we get a solution to to x_{k-j} or one of its reductions.

2.4 Easily certifiable integrality gaps

Theorem 2 presents a family of integer programs (call this family F_β), such that for every IP in F_β the LP-relaxation has an integrality gap no worse than β (and furthermore, no polynomial time "rounding" algorithm ensures approximation within a ratio better than α, unless TFNP is in FP). Recall (see proof of Proposition 14) that the IPs in F_β are derived from some relation $R \in \mathsf{TFNP}$. We may assume that the Turing machine T (and hence also the circuits C_n) associated with R is publicly known. In this case, for every $IP \in F_\beta$ there is a polynomial size witness (which includes the input x' for R and the chain or reductions leading to the IP) that certifies that indeed the IP is in F_β, and hence that the integrality gap is no worse than β. However, this does not necessarily imply that given an IP in F_β, finding the associated witness for being in F_β is a computationally easy task (this is not required in the statement of Theorem 2). We sketch below how one can modify the proofs in this manuscript so that the resulting family F_β is such that finding witnesses for $IP \in F_\beta$ becomes an easy computational task.

As observed in the proof of Theorem 2, given an IP as in Theorem 2, it is easy to decompose it into IP_1 and IP_2, and to separate the objective function into a part that depends on IP_1 and a part that depends on IP_2. The form of IP_2 makes it self-evident that it represents a 3CNF formula (that can be explicitly reconstructed from the constraints of IP_2). However, without a-priori knowing the instance x', the task of determining that IP_1 is an IP that is derived by a reduction from an instance of R might be difficult. To make this task easy, one can slightly modify the proof of Proposition 14. Namely, first reduce the circuit C_n (rather than the simplified circuit $C_n(x')$) to an IP. Thereafter, given x', for each input bit i the reduction adds either the constraint $x_i = 0$ or the constraint $x_i = 1$, depending on the value of bit i in x'. Finally, the reduction names the variables in the order in which they appear in the circuit C_n. (Alternatively, if we do not wish names of variables to convey information, there are other tricks of associating an index i with a variable z, e.g., by adding a trivially satisfiable constraint $z \leq i$.) Given an IP constructed as above, one can easily reconstruct x' and the circuit C_n, and consequently certify that the integrality gap is no worse than β.

Acknowledgements

Work supported in part by the Israel Science Foundation (grant No. 621/12) and by the I-CORE Program of the Planning and Budgeting Committee and The Israel Science Foundation (grant No. 4/11).

References

[1] Nikhil Bansal. Semidefinite optimization in discrepancy theory. Math. Program. 134(1): 5–22 (2012).

[2] Xi Chen, Xiaotie Deng. Settling the Complexity of Two-Player Nash Equilibrium. FOCS 2006: 261–272.

[3] Uriel Feige. On Estimation Algorithms vs Approximation Algorithms. FSTTCS 2008: 357–363.

[4] Bernhard Haeupler, Barna Saha, Aravind Srinivasan. New Constructive Aspects of the Lovasz Local Lemma. J. ACM 58(6): 28 (2011).

[5] Johan Hastad. Some optimal inapproximability results. J. ACM 48(4): 798–859 (2001).

[6] Dorit Hochbaum (Ed.). *Approximation Algorithms for NP-hard Problems.* PWS Publishing Company, 1997.

[7] David S. Johnson, Christos H. Papadimitriou, Mihalis Yannakakis. How Easy is Local Search? J. Comput. Syst. Sci. 37(1): 79–100 (1988).

[8] Shachar Lovett, Raghu Meka. Constructive Discrepancy Minimization by Walking on the Edges. FOCS 2012: 61–67.

[9] Nimrod Megiddo and Christos H. Papadimitriou. On total functions, existence theorems, and computational complexity, Theoretical Computer Science 81(2):317–324, 1991.

[10] Robin A. Moser, Gabor Tardos. A constructive proof of the general Lovasz Local Lemma. J. ACM 57(2) (2010).

[11] Christos H. Papadimitriou. On Graph-Theoretic Lemmata and Complexity Classes (Extended Abstract) FOCS 1990: 794–801.

[12] Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence, Journal of Computer and System Sciences 48(3):498–532, 1994.

[13] Lukas Polacek, Ola Svensson. Quasi-polynomial Local Search for Restricted Max-Min Fair Allocation. ICALP (1) 2012: 726–737.

[14] David P. Williamson, David B. Shmoys. The Design of Approximation Algorithms. Cambridge University Press 2011.

[15] Vijay Vazirani. *Approximation Algorithms.* Springer 2001.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de