Lecture 10: Linear Programming and Randomised Rounding

"Brand new LP just dropped!"

Over your previous years and courses, you have seen a range of powerful algorithm design techniques with which to attack algorithmic problems. Once you have formulated the task you want to solve, you have by now an array of tools to try and solve it: greedy algorithms, divide-and-conquer, dynamic programming... Linear Programming (LP for short) is another one, very powerful. We will only scratch the surface here, but what it does is quite simple to state:

> Maximize a linear function subject to linear inequality constraints on variables $x_1, \ldots, x_n$ of interest.

In its most general form, this can be rephrased as follows:

You can do minimisation tasks instead of maximisation, of course. Can you see how?

$$\begin{aligned} &\text{maximise } c^\intercal x \\ &\text{subject to } Ax \leq b \qquad\qquad x \geq 0 \end{aligned}$$

where the inequalities are to be taken coordinate-wise; $x \in \mathbb{R}^n$ is the set of variables (encoding the solution), and $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, and $b \in \mathbb{R}^m$ encode the task to be solved: c the objective function, and A, b the constraints. This may look a little daunting in this form, so here is a less compressed version:

$$\begin{aligned} &\text{maximise } \sum_{i=1}^{n} c_i x_i \\ &\text{subject to} \\ &\qquad \sum_{i=1}^{n} A_{ji} x_i \leq b_j, \qquad 1 \leq j \leq m \\ &\qquad\qquad x_i \geq 0, \qquad 1 \leq i \leq n \end{aligned}$$

We will see examples through the chapter, but here are the key things to keep in mind regarding Linear Programming:

- LP is P-complete: every problem that has a polynomial-time algorithm can be solved by (some) linear program.
- LPs can be solved efficiently *in theory*: every linear program can be solved (to arbitrary precision) in time polynomial in n, m (or, more precisely, in the size of the representation of A, b, c, and desired accuracy.)
- LPs can be solved efficiently *in practice*: there are actual algorithms which solve LPs very quickly, and you can use them.
- There is a *very rich theory* on Linear Programming, enough to fill a whole course: we will not touch on most of it.

For instance, COMP4530/5530: Discrete Optimisation. Consider taking that course!

So, for the rest of this chapter, whenever we end up with an LP (on a polynomial number of variables n and constraints m), we will simply wave a magic wand and say "this can be efficiently solved to get an optimal solution," without caring too much about *how*.

LPs are very powerful, and a wonderful arrow in our algorithmic quiver, allowing us in principle to solve in a systematic way every problem in P.[44] The issue, of course, is that often we want to solve problems that are *not* known to be in P: problems which, by definition, we do not know how to express as an LP with a polynomial number of variables and constraints. But we would still like to solve these approximately! *Can LPs still help somehow?*

[44] This is not saying we *should*. There may be faster algorithms than writing and solving the corresponding LP!

Specifically, as in the chapter on streaming (but for different reasons), given a hard computational task $\mathcal{T}$ we would like to obtain an α-approximation to an optimal solution to $\mathcal{T}$, for some approximation factor $0 < \alpha \leq 1$: that is, giving an instance I of $\mathcal{T}$, we want to output a solution $S = S(I)$ whose value satisfies

$$\alpha \cdot \textsc{opt}(I) \leq \textsc{val}(S) \leq \textsc{opt}(I) \tag{63}$$

For maximisation problems. For minimisation, that would be $\textsc{opt}(I) \leq \textsc{val}(S) \leq \alpha \cdot \textsc{opt}(I)$, for $\alpha \geq 1$.

(with high probability, or in expectation, if our algorithm is randomised), where $\textsc{opt}(I)$ denotes the optimal value: the best achievable by any solution for instance I.

Let's forget about LPs.

One standard approach is to first, essentially, shrug and formulate our problem as something which is *not* an LP, but instead includes some stronger constraints that are not linear. That is, instead of constraints on our variables like $x_i \geq 0$ we allow constraints of the form "x_i must be an integer." (Most often, $x_i \in \{0, 1\}$.)

Here is an example, for a problem that is, actually, in P, one we have seen before: st-Min-Cut. To interpret this: we have $|V| + |E|$ variables. The x_e's indicate whether edge e is part of the cut, while

$$\begin{aligned}
&\text{maximise } -\sum_{e\in E} c_e x_e \\
&\text{subject to} \\
&\qquad y_s = 0 \\
&\qquad y_t = 1 \\
&\qquad y_v \le y_u + x_e, \qquad \forall e = (u,v) \in E \\
&\quad x_e, y_v \in \{0,1\} \qquad \forall e \in E, v \in V
\end{aligned}$$

Figure 13: MIN-CUT, on a directed graph $G = (V, E)$ with edge weights $c\colon E \to \mathbb{R}_+$ and source and sink vertices $s, t \in V$, formulated as an ILP.

the y_v's indicate whether vertex v is in the same connected component as the sink t. The goal is to select values for the variables in order to minimise the weight of the cut, $\sum_{e\in E} c_e x_e$. So if we could solve this optimally, we would have a solution to our st-MIN-CUT instance!

As we will see, switching from LPs to ILPs comes at a significant advantage: we can encode many more problems as Integer Linear Programs (ILPs), including NP-Hard ones. This also comes at a significant cost: contrary to LPs, we just don't know how to solve ILPs efficiently in general!

Integer Linear Program (ILP)

So *what is the point of all this?* We know how to solve LPs very fast, but they may not capture the problems we care to solve. ILPs might, but we don't know how to solve them efficiently. This is even more dire given the example above: we *have* very efficient algorithms for st-MIN-CUT. But now that we formulated MIN-CUT as an ILP, we just... don't know how to solve it that way?

Let's not forget about LPs.

Here comes the key insight: once we have formulated a problem as an ILP, we can *relax* its constraints to convert it into an LP *which we then can solve efficiently*. This gives us a solution to a different problem (since we changed the constraints), but sometimes, if we are lucky, from this solution to the *relaxed* problem we can extract a solution to the *original* problem, and still can say something interesting about its value (quality).

"LP Relaxation"

Relaxing the problem basically means converting all the "hard" integer constraints into continuous, linear ones: *e.g.*, for the MIN-CUT problem, relaxing the ILP above yields the LP given in Fig. 14. We can then solve this, giving us a solution S_{LP} to our problem. We then have the nice, "obvious" fact:

Fact 45.1. *Let* OPT_{ILP} *be the optimal value of a solution to an ILP (maximisation problem), and* OPT_{LP} *be the optimal value of a solution to its LP relaxation. Then*

$$\text{OPT}_{\text{ILP}} \le \text{OPT}_{\text{LP}}\,.$$

(For a minimisation problem, the inequality is reversed.)

Figure 14: The LP relaxation to the previous MIN-CUT ILP relaxation.

$$\begin{aligned} \text{maximise } & -\sum_{e \in E} c_e x_e \\ \text{subject to} & \\ & y_s = 0 \\ & y_t = 1 \\ & y_v \leq y_u + x_e, \qquad \forall e = (u,v) \in E \\ & x_e, y_v \in [0,1] \qquad \forall e \in E, v \in V \end{aligned}$$

Of course, this will not in general be a solution to the original problem (here, st-MIN-CUT), because, well, what does it mean for a vertex to have $y_v = 0.5786$? Which side of the cut is vertex v?

Rounding! The next key insight is that we can often *extract* a solution to the ILP from the (optimal) solution to the LP. Now, we will have to lose something in the process: either the solution is only a valid solution with high probability (with small probability, some of the constraints will be violated), or the value of the resulting solution is not quite optimal. Here, we will focus on the latter. Going back to our MIN-CUT example: we solved the LP relaxation, getting an optimal solution

Otherwise, we would know how to solve the ILP! Possibly proving P=NP along the way.

$$(x^*, y^*) \in [0,1]^{|E|+|V|}$$

to the LP, with value OPT_{LP}. We want to get a valid solution to our graph problem, so getting a solution $y \in \{0,1\}^{|V|}$ from this for which, hopefully, $\text{VAL}(y) \approx \text{OPT}_{\text{LP}}$.

A very natural idea: let's *round the coordinates* of y^*! Rounding things in $[0,1]$ will give us things in $\{0,1\}$, and that seems to fit the bill. This works for *some* problems, and is called *deterministic rounding*. The hard part, however, is then to be able to argue anything about $\text{VAL}(y)$: for MIN-CUT for instance, if all coordinates of y^* are say 0.50001 (except of course y_s), then we round them all to 1, and we include *all* vertices except s in our cut. Maybe not a good idea.

Check your understanding: why are we looking at y, and not x (say, setting x_e to 1 with probability x_e^*)?

But what about *randomised* rounding then? One issue with the above rounding is that we had a *deterministic* threshold, $1/2$, which did not allow us to say much about the result, and in particular did not let us leverage any guarantee provided by the LP we just solved. So, deterministic threshold: not very useful. Maybe we can pick our threshold *randomly* then?

Algorithm 21: Randomised rounding for the LP relaxation of MIN-CUT.

1: Pick τ in $(0,1)$ uniformly at random.
2: **for all** $v \in V$ **do**
3: Set $y_v = 1$ if $y_v^* > \tau$, 0 otherwise
4: **return** y.

That's all. *Does it work?* Clearly, this returns a valid cut, as $y \in \{0,1\}^{|V|}$ with $y_s = 0, y_t = 1$. Can we say anything about the (expected) value of the ? As it turns out, yes! Using the LP we solved as a guide.

Theorem 46. *The cut y returned by Algorithm 21 satisfies* $\mathbb{E}[\textsc{val}(y)] = \textsc{opt}_{\text{ILP}}$.

Proof. By linearity of expectation,

$$\mathbb{E}[\textsc{val}(y)] = -\sum_{e \in E} c_e \mathbb{E}[\mathbb{1}_{e \text{ is cut}}] = -\sum_{e=(u,v)\in E} c_e \Pr[\, y_u = 0, y_v = 1 \,]$$

We want to relate this to $-\sum_{e\in E} c_e x^*_e$, since this is what we know is the optimal value $\textsc{opt}_{\text{LP}}$ of the LP. But, for any $e = (u,v) \in E$, using the third constraint of the LP, $y^*_v - y^*_u \leq x^*_e$, and so

$$\Pr[\, y_u = 0, y_v = 1 \,] = \Pr[\, y^*_u \leq \tau < y^*_v \,] \leq x^*_e$$

from which

$$-\mathbb{E}[\textsc{val}(y)] \leq \sum_{e=(u,v)\in E} c_e x^*_e = -\textsc{opt}_{\text{LP}} \leq -\textsc{opt}_{\text{ILP}}$$

the last inequality from Fact 45.1. In expectation, the solution we get is at least as good as the minimum cut value: since it cannot be *better* (no valid cut can be better than the minimum cut!), it *is* the minimum cut value. □

Note that while the above gives a guarantee on the *expected* value of the min-cut, we actually implicitly proved something much stronger: namely, that the solution returned *always* has optimal value (not just in expectation). To see why, observe that the above theorem states, since $\textsc{opt}_{\text{ILP}} = \textsc{opt}_{\text{MinCut}}$, that

$$\mathbb{E}[\textsc{val}(y)] = \textsc{opt}_{\text{MinCut}}\,.$$

but clearly, since the returned solution y is *some* cut, we also have $\textsc{val}(y) \geq \textsc{opt}_{\text{MinCut}}$ for all possible y returned. But then this means we must have $\textsc{val}(y) = \textsc{opt}_{\text{MinCut}}$ with probability one!

Check why: $X := \textsc{val}(y) - \textsc{opt}_{\text{MinCut}}$ is a non-negative random variable with expectation equal to 0: it must be equal to 0 with probability 1.

This is nice! But we used a big hammer (ILP, LP relaxation, then randomised rounding) in order to solve a problem we already knew how to solve deterministically *via* Max-Flow, and it's not even clear the new approach is faster. This was very good for the sake of illustrating the ideas, but *surely*, there has to be more compelling? To see one, let us turn to a *bona fide* NP-Hard problem, Max-SAT.

Approximation algorithm for Max-SAT

In the *maximum satisfiability problem* (Max-SAT), we have n Boolean variables $x_1, \ldots, x_n \in \{0,1\}$, grouped into m *clauses* $C_1, \ldots, C_m$. Each clause is a *disjunction* of the variables, that is, of the form

$$C_j = x_{i_1} \vee \neg x_{i_2} \vee \cdots \vee x_{i_\ell}$$

(a logical OR of *literals*, where a literal is either a variable x_i or its negation $\neg x_i$). A clause is *satisfied* if it evaluates to 1, that is, if at least one of the literals in the clause is 1. The MAX-SAT problem asks, given such a formula $\phi = (C_1, \dots, C_m)$, to assign values to all n variables in order to maximise the number of satisfied clauses, *i.e.*,

$$\text{VAL}_\phi(x_1, \dots, x_n) = \sum_{j=1}^{m} \mathbb{1}_{C_j \text{ is satisfied}}$$

All we are going to talk about generalises to the weighted version, where clause C_j has a weight $w_j \geq 0$ and the goal is to maximise $\sum_{j=1}^{m} w_j \mathbb{1}_{\{C_j \text{ is satisfied}\}}$. Check it!

Without loss of generality, we assume that (1) all m clauses are distinct, (2) x_i and $\neg x_i$ do not appear both in any given clause (as this makes it automatically satisfied), and (3) each literal appears at most once in each clause (no repetition, as they are useless). The *length* of a clause C_j is the number of literals in the clause, denoted $\ell_j = |C_j|$.

Fact 46.1. MAX-SAT *is NP-Hard. (Even deciding whether* $\text{OPT}(\phi) = m$ *is NP-Complete.)*

But can we *approximate* $\text{OPT}(\phi)$? As it turns out, getting *some* approximation is not too difficult:

Theorem 47. *The "obvious" randomised algorithm which sets each variable x_i independently and uniformly at random gives, in expectation, a $\frac{1}{2}$-approximation for* MAX-SAT.

Proof. Consider a fixed clause C_j. The probability that C_j is *not* satisfied is the probability to set every single one of the ℓ_j literals the wrong way, which is

$$\frac{1}{2^{\ell_j}}$$

and so, by linearity of expectation, and as $\ell_j \geq 1$ for all $1 \leq j \leq m$,

$$\mathbb{E}\left[\text{VAL}_\phi(x)\right] = \sum_{j=1}^{m} \left(1 - \frac{1}{2^{\ell_j}}\right) \geq \frac{1}{2}m \geq \frac{1}{2}\text{OPT}(\phi)\,. \tag{64}$$

proving the result. □

Interestingly, one nice feature is that this gives a better guarantee for "long clauses", those for which $\ell_j \geq 2$. For instance, for MAX-E3SAT, where each clause has *exactly* 3 literals, we get a $\frac{7}{8}$-approximation (since $1 - 1/2^{\ell_j} = \frac{7}{8}$ for every j)!

Again, this is fine, but (1) a $(1/2)$-approximation is not that exciting, and (2) there is no LP in there! Now that the warmup is over, let us formulate MAX-SAT as an ILP. Besides the n variables $y_1, \dots, y_n$ (corresponding directly to the n Boolean variables $x_1, \dots, x_n$), we will have m additional variables, one per clause, where z_j indicates whether C_j is satisfied: We then need to show the following:

Lemma 47.1. *The optimal value of the ILP is equal to* $\text{OPT}(\phi)$.

Sketch. To check: if $x \in \{0,1\}^n$ is an optimal solution to the MAX-SAT problem on input $\phi = (C_1, \dots, C_m)$, then one can extract from

Figure 15: MAX-SAT, formulated as an ILP.

$$\begin{aligned}
&\text{maximise} && \sum_{j=1}^{m} z_j \\
&\text{subject to} \\
&&\sum_{i:x_i \in C_j} y_i + \sum_{i:\neg x_i \in C_j} (1 - y_i) \geq z_j && \forall 1 \leq j \leq m \\
&& y_i \in \{0,1\} && \forall 1 \leq i \leq n \\
&& z_j \in \{0,1\} && \forall 1 \leq j \leq m
\end{aligned}$$

it a valid solution to the ILP, with the same value. Conversely, from an *optimal* solution to the ILP, one can obtain a solution to MAX-SAT with the same value. □

As before, we don't know how to solve this: so we go for the LP relaxation. After solving (optimally) this LP relaxation in time

Figure 16: MAX-SAT, LP relaxation to the above ILP.

$$\begin{aligned}
&\text{maximise} && \sum_{j=1}^{m} z_j \\
&\text{subject to} \\
&&\sum_{i:x_i \in C_j} y_i + \sum_{i:\neg x_i \in C_j} (1 - y_i) \geq z_j && \forall 1 \leq j \leq m \\
&& 0 \leq y_i \leq 1 && \forall 1 \leq i \leq n \\
&& 0 \leq z_j \leq 1 && \forall 1 \leq j \leq m
\end{aligned}$$

polynomial in n and m, we get a solution y^*, z^* optimal for this LP. How do we round this to get a (valid) solution x for the ILP (equivalently, for MAX-SAT) about which we can prove something? Here's another key idea: we want values $x_i \in \{0,1\}$ but we are given $y_i \in [0,1]$. The good thing is, there is a very natural interpretation to values in $[0,1]$: seeing them as *probabilities*. This suggests the following randomised rounding scheme:

Algorithm 22: Randomised rounding of the LP relaxation for MAX-SAT.

Input: Instance $\phi = (C_1, \ldots, C_m)$ of MAX-SAT on n variables
1: Solve the LP relaxation (Fig. 16), getting solution (y^*, z^*).
2: **for all** $1 \leq i \leq n$ **do**
3: Set $x_i = 1$ with probability y_i^*, independently of others.
4: **return** x.

This is quite simple, and produces a valid assignment of the Boolean variables $x_1, \ldots, x_n$. How well does it fare in terms of clauses satisfied? As it turns out, not badly at all!

Theorem 48. *The randomised rounding given in Algorithm 22 gives, in expectation, a* $(1 - \frac{1}{e})$*-approximation for* MAX-SAT.

$1 - 1/e \approx 0.632$: better than $1/2$!

Proof. As before, we need to use what the LP we solved did, as a guide to analyse the expected value of our solution. What we can write is

$$\mathbb{E}\big[\textsc{val}_\phi(x)\big] = \sum_{j=1}^{m} \Pr\big[\, C_j \text{ satisfied} \,\big] = \sum_{j=1}^{m} \big(1 - \Pr\big[\, C_j \text{ not satisfied} \,\big]\big)$$

By independence of our assignments of the x_i's, and since each variable appears (negated or not) at most once per clause, we also have that, for every j,

$$\begin{aligned}\Pr\big[\, C_j \text{ not satisfied} \,\big] &= \prod_{i:x_i \in C_j} \Pr[\, x_i = 0 \,] \cdot \prod_{i:\neg x_i \in C_j} \Pr[\, x_i = 1 \,] \\ &= \prod_{i:x_i \in C_j} (1 - y_i^*) \cdot \prod_{i:\neg x_i \in C_j} y_i^* \qquad (65)\end{aligned}$$

This is a product, but what the LP gives us a handle on is a sum: from the constraints, what we know is that, for every j,

$$\sum_{i:x_i \in C_j} y_i + \sum_{i:\neg x_i \in C_j} (1 - y_i) \geq z_j$$

To relate products to sums (or, rather, to averages), one very handy tool is the *inequality of arithmetic and geometric means* (AM–GM inequality):

Another life-saver: the AM–GM inequality.

Fact 48.1. *For any $a_1, \dots, a_k \geq 0$,*

$$\sqrt[k]{a_1 a_2 \cdots a_k} \leq \frac{a_1 + a_2 + \cdots + a_k}{k}$$

Applying this to (65) gives us

$$\begin{aligned}\Pr\big[\, C_j \text{ not satisfied} \,\big] &= \prod_{i:x_i \in C_j} (1 - y_i^*) \cdot \prod_{i:\neg x_i \in C_j} y_i^* \\ &\leq \left(\frac{\sum_{i:x_i \in C_j} (1 - y_i^*) + \sum_{i:\neg x_i \in C_j} y_i^*}{\ell_j} \right)^{\ell_j} \\ &= \left(1 - \frac{\sum_{i:x_i \in C_j} y_i^* + \sum_{i:\neg x_i \in C_j} (1 - y_i^*)}{\ell_j} \right)^{\ell_j} \\ &\leq \left(1 - \frac{z_j^*}{\ell_j} \right)^{\ell_j} \qquad \text{(By the LP constraint)}\end{aligned}$$

and so, recalling where we came from,

$$\mathbb{E}\big[\textsc{val}_\phi(x)\big] \geq \sum_{j=1}^{m} \left(1 - \left(1 - \frac{z_j^*}{\ell_j} \right)^{\ell_j} \right) \qquad (66)$$

We are almost there! We want to relate this to what we *know* is the optimal value of the LP, $\sum_{j=1}^{m} z_j^*$. To do so, observe that for any $\ell \geq 1$ the function

$$f(z) = 1 - \left(1 - \frac{z}{\ell} \right)^{\ell}, \qquad z \in [0, 1]$$

is concave, and as a result is above its linear interpolation (see Fig. 17):

Exercise: prove it!

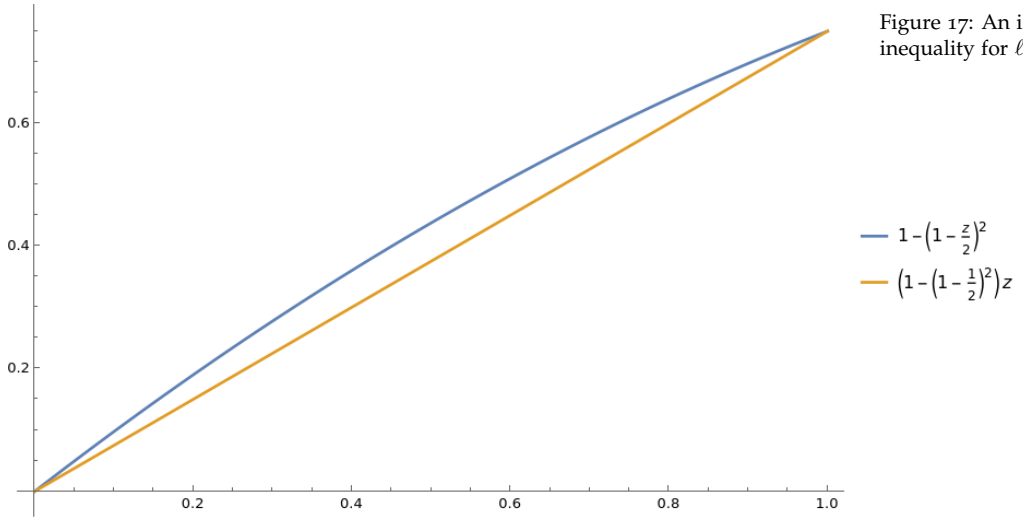

Figure 17: An illustration of the inequality for $\ell = 2$.

$$f(z) \geq \left(1 - \left(1 - \frac{1}{\ell}\right)^{\ell}\right) z, \qquad z \in [0,1]$$

Using this, we finally get

We invoke the "fact" that $\sup_{\ell \geq 1} \left(1 - \frac{1}{\ell_j}\right)^{\ell_j} = 1/e$.

$$\begin{aligned}
\mathbb{E}\left[\textsc{val}_{\phi}(x)\right] &\geq \sum_{j=1}^{m} \left(1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j}\right) \cdot z_j^* \qquad (67)\\
&\geq \inf_{\ell \geq 1} \left(1 - \left(1 - \frac{1}{\ell}\right)^{\ell}\right) \cdot \sum_{j=1}^{m} z_j^* \\
&= \left(1 - \frac{1}{e}\right) \textsc{opt}_{\text{LP}} \\
&\geq \left(1 - \frac{1}{e}\right) \textsc{opt}_{\text{ILP}}
\end{aligned}$$

concluding the proof. □

Are we done? We started by showing a simple randomised approach giving an expected $1/2$-approximation to Max-SAT. By spending more time, effort, and relaxing (an ILP), we obtained an efficient randomised algorithm, less simple but giving a better guarantee: an expected 0.632-approximation. *Can we do better?*

Surprisingly, yes: we can get a $3/4$-approximation, by combining the two! This is quite unexpected, as combining two algorithms will give a better result than *both*. The crucial insight is that, as remarked before, the naive randomised approach fares very well on *long* clauses. What about the LP-relaxation one? Roughly speaking, its expected approximation guarantee on a clause of length ℓ_j is

$$\left(1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j}\right)$$

which is better for short clauses! (As you can see, for $\ell_j = 1$ this takes value 1, for $\ell_j = 2$ it is $3/4 \ldots$ We may hope that choosing the best

of the two solutions (one better when clauses are typically long, the other better when they are typically short) could be beneficial.

Theorem 49. *The "best-of-two" approach which runs both the naive randomised algorithm of Theorem 47 and the randomised rounding of Theorem 48 gives, in expectation, a* $3/4$*-approximation for* Max-SAT.

Proof. The proof is quite simple: denote by x, x' the two solutions returned. Then, since the max is at least the average,

$$\begin{aligned}
\mathbb{E}\left[\max\left(\textsc{val}_\phi(x), \textsc{val}_\phi(x')\right)\right] &\geq \frac{1}{2}\mathbb{E}\left[\textsc{val}_\phi(x) + \textsc{val}_\phi(x')\right] \\
&\geq \frac{1}{2}\sum_{j=1}^{m}\left(\left(1-\frac{1}{2^{\ell_j}}\right) + \left(1-\left(1-\frac{1}{\ell_j}\right)^{\ell_j}\right)\cdot z_j^*\right) \\
&\qquad \text{(By (64) and (67))} \\
&\geq \frac{1}{2}\sum_{j=1}^{m}\left(\left(1-\frac{1}{2^{\ell_j}}\right) + \left(1-\left(1-\frac{1}{\ell_j}\right)^{\ell_j}\right)\right)\cdot z_j^* \\
&\qquad \text{(as } z_j^* \leq 1\text{)} \\
&\geq \frac{3}{4}\sum_{j=1}^{m} z_j^* \qquad (\star) \\
&\geq \frac{3}{4}\textsc{opt}_{\text{ILP}}
\end{aligned}$$

where the inequality $(\star)$ follows from the (somewhat technical) claim that, for every $x \geq 2$ *and* for $x = 1$, we have (see Fig. 18)

$$\left(1-\frac{1}{2^x}\right) + \left(1-\left(1-\frac{1}{x}\right)^x\right) \geq \frac{3}{2}.$$

This concludes the proof, and the chapter. □

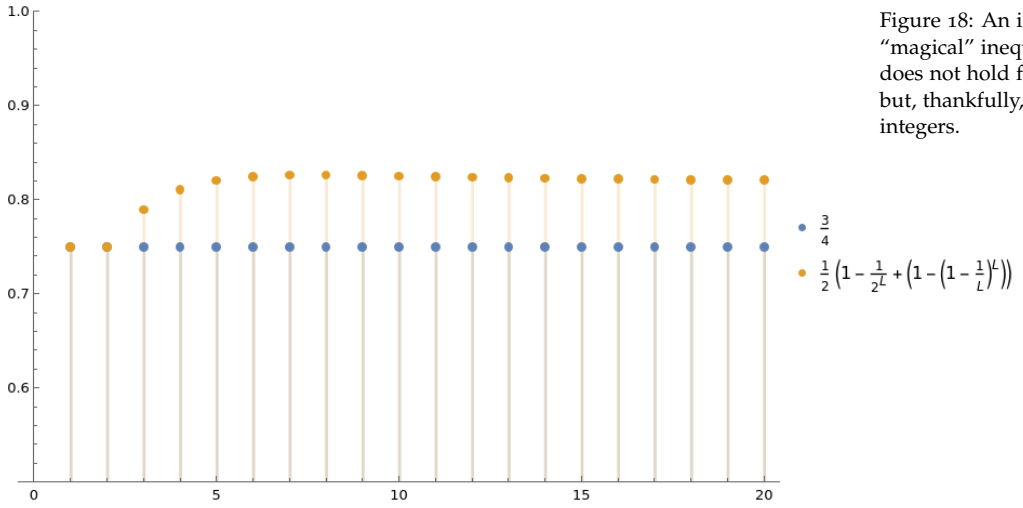

Figure 18: An illustration of the last, "magical" inequality. Note that it does not hold for values in $(1,2)$, but, thankfully, we only need it for integers.