A Short Guide to Approximation Preserving Reductions

Pierluigi Crescenzi*
Dipartimento di Scienze dell'informazione
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
E-mail: piluc@dsi.uniroma1.it

Abstract

Comparing the complexity of different combinatorial optimization problems has been an extremely active research area during the last 23 years. This has led to the definition of several approximation preserving reducibilities and to the development of powerful reduction techniques. We first review the main approximation preserving reducibilities that have appeared in the literature and suggest which one of them should be used. Successively, we give some hints on how to prove new non-approximability results by emphasizing the most interesting techniques among the new ones that have been developed in the last few years.

1. Introduction

> What is it that makes algorithms for different problems behave in the same way? Is there some stronger kind of reducibility than the simple polynomial reducibility that will explain these results, or are they due to some structural similarity between the problems as we define them?
>
> David S. Johnson [26]

> ... ce sont rarement les réponses qui apportent la vérité, mais l'enchaînement des questions.
>
> Daniel Pennac [38]

Suppose that you have decided to buy a new computer. The first thing you will probably realize is the incredibly high number of possibilities you have, ranging from cheap personal computers to extremely expensive workstations. This should lead you to change your original question: "Which computer should I buy?" to the more intimate question: "What am I going to do with the computer?". Clearly, one possible consequence of this change of focus is the decision of not buying a computer at all. But let us assume that, after all, you really need one. After carefully analyzing your needs, you finally decide to buy the RTM 9000 personal computer.

Eventually, you will get it, unpack it, and put it on the table just in front you. The next question will then be "How do I use it?". The answer to this question may depend on the computer itself: the so-called user-friendly computers should allow you to learn to use them without any further help while most of the others will force you to read dozens and dozens of pages on quite cryptic user manuals. Certainly, after a while you will learn how to use the RTM 9000 even though you will also realize that in the meanwhile much better machines have been produced. This will likely lead you to the ultimate question "Should I buy a new computer?".

A quite similar situation can arise when you decide to prove that a given optimization problem A is not approximable. As stated in [2], "the first step in proving an inapproximability result for a given problem is to check whether it is already known to be inapproximable. For instance, the problem might be listed in [14]." If you are lucky, you will find the answer to your question with, basically, no effort (but, unfortunately, with no publication either ...). Otherwise, you can either try to prove that approximating problem A is NP-hard or try to compare problem A with any of the optimization problems for which inapproximability results are already known.

After examining the literature, you will immediately realize that a fundamental tool in order to follow the second alternative is the notion of approximation preserving reducibility which should allow you to compare the complexity of approximating A to that of some other optimiza-

*Starting from November 1, 1997, the author's new affiliation will be: Dipartimento di Sistemi ed Informatica, Università di Firenze, Via Cesare Lombroso 6/17, 50134 Firenze, Italy (e-mail: piluc@dsi2.dsi.unifi.it).

1093-0159/97 $10.00 © 1997 IEEE

tion problem. Different from proving NP-completeness (in which case the only reducibility to be used is the many-to-one polynomial-time reducibility), you will also discover that many approximation preserving reducibilities can be used, ranging from very strict kinds of reducibilities to extremely general ones.

The first goal of this paper is to briefly review the main reducibilities that have been defined in the last 20 years organizing them into a taxonomy (a similar overview of approximation preserving reducibilities that have been defined in the 80s is contained in [11]). After examining the differences between these reducibilities, we will suggest you concentrate your attention on three of them, the L-reducibility, the AP-reducibility and the PTAS-reducibility. The choice between these three reducibilities will then depend mainly on the following question: "What am I going to do with the reducibility?". Most of the time, the answer will be: "To prove a non-approximability result" in which case the L-reducibility will likely suffice. Sometimes, instead, the answer will be: "To prove a completeness result" in which case you should seriously consider the possibility of switching either to the AP-reducibility or to the PTAS-reducibility.

Once you have decided which reducibility you are going to use, the next question will be: "How do I develop an approximation preserving reduction?". This is a much harder question and, unfortunately, there is no user-manual (not even a cryptic one) that can explain the secrets of proving these kinds of results.

The second goal of this paper is to give some guidelines that, we hope, can be usefully followed in order to obtain the desired result. To this aim, we will describe two of the most interesting techniques that have been developed in the last few years in order to obtain non-approximability results: *expansion* and *randomization*. We believe that these techniques are general enough to deserve a special treatment. Paraphrasing [20], this will not be a classification of all non-approximability results. Our main intent will be to illustrate two ways of thinking about approximation preserving reduction proofs that were not explicitly present in previously known NP-completeness proofs.

Clearly, it may be still possible that your attempts to prove the non-approximability of A fail: do not forget that it is possible that A is very well approximable (or even solvable in polynomial-time). In the meanwhile, more powerful approximation preserving reducibilities could have been defined by somebody else and you could ask yourself: "Should I use this new reducibility?". Even more. You could ask yourself: "Should I define a new approximation preserving reducibility?". Indeed, this has been the approach followed by several authors (including this guide's one) in the last ten years: this is the reason why so many reducibilities are now available "on the market"!

Organization of the guide The paper is organized as follows. In Section 2, we give the basic definitions regarding optimization problems and approximation algorithms. In Section 3, we give the general definition scheme of an approximation preserving reducibility, and a taxonomy of existing reducibilities is presented. At the end of the section, we recommend three of these reducibilities and try to suggest when they should be used. Finally, in Section 4 we outline our approximation preserving reduction guide emphasizing the role of expander graphs and of randomization as new techniques useful to obtain reductions.

2. Basic definitions

We assume the reader to be familiar with the basic concepts of computational complexity theory as presented in one of the available text books (see, for example, [8, 10, 20, 35]).

We now give some standard definitions in the field of optimization and approximation theory. For a more detailed exposition of this theory we refer the reader to [5, 24]. A preliminary version of some chapters of [4] is also available on request to one of the authors.

Definition 1 *An* NP optimization problem A *is a fourtuple* $(I, \mathsf{sol}, \mathsf{m}, \mathrm{type})$ *such that*

1. I *is the set of the* instances *of* A *and it is recognizable in polynomial time.*
2. *Given an instance* x *of* I, $\mathsf{sol}(x)$ *denotes the set of* feasible solutions *of* x. *These solutions are short, that is, a polynomial* p *exists such that, for any* $y \in \mathsf{sol}(x)$, $|y| \leq p(|x|)$. *Moreover, for any* x *and for any* y *with* $|y| \leq p(|x|)$, *it is decidable in polynomial time whether* $y \in \mathsf{sol}(x)$.
3. *Given an instance* x *and a feasible solution* y *of* x, $\mathsf{m}(x, y)$ *denotes the positive integer* measure *of* y *(often also called the value of* y*). The function* m *is computable in polynomial time and is also called the* objective *function.*
4. $\mathrm{type} \in \{\max, \min\}$.

The goal of an NP optimization problem with respect to an instance x is to find an *optimum solution*, that is, a feasible solution y such that

$$\mathsf{m}(x, y) = \mathrm{type}\{\mathsf{m}(x, y') \ : \ y' \in \mathsf{sol}(x)\}.$$

In the following opt will denote the function mapping an instance x to the measure of an optimum solution. The *class* NPO is the set of all NP optimization problems.

In the following we will mainly refer to satisfiability optimization problems. In particular, we will make use of the following optimization problems.

MAX SAT

INSTANCE: Set X of variables, collection φ of disjunctive clauses of literals, where a literal is a variable or a negated variable in X.
SOLUTION: A truth assignment for X.
MEASURE: Number of clauses satisfied by the truth assignment.

MAX kSAT

INSTANCE: Set X of variables, collection φ of disjunctive clauses of at most k literals, where a literal is a variable or a negated variable in X (k is a constant, $k \geq 2$).
SOLUTION: A truth assignment for X.
MEASURE: Number of clauses satisfied by the truth assignment.

Definition 2 *Let A be an* NPO *problem. Given an instance x and a feasible solution y of x, we define the* performance ratio of y with respect to x *as*

$$R(x,y) = \max\left\{\frac{m(x,y)}{\mathsf{opt}(x)}, \frac{\mathsf{opt}(x)}{m(x,y)}\right\}.$$

The performance ratio is always a number greater than or equal to 1 and is as close to 1 as the value of the feasible solution is close to the optimum value.

Definition 3 *Let A be an* NPO *problem and let T be an algorithm that, for any instance x of A, returns a feasible solution $T(x)$ in polynomial time. Given a rational $r > 1$, we say that T is an* r-approximation algorithm for A *if the performance ratio of the feasible solution $T(x)$ with respect to x verifies the following inequality:*

$$R(x, T(x)) \leq r.$$

Definition 4 *An* NPO *problem A belongs to the* class APX *if an r-approximation algorithm T for A exists, for some rational $r > 1$.*

Definition 5 *An* NPO *problem A belongs to the* class PTAS *if an algorithm T exists such that, for any fixed rational $r > 1$, $T(\cdot, r)$ is an r-approximation algorithm for A.*

Clearly, the following inclusions hold:

$$\mathsf{PTAS} \subseteq \mathsf{APX} \subseteq \mathsf{NPO}.$$

The major open problem in the theory of approximation complexity is whether the inclusions among these three classes are strict. It is easy to see that this is the case if and only if $\mathsf{P} \neq \mathsf{NP}$, so that solving this problem seems to be very difficult. Parallelizing the development of the theory of NP-completeness, we can then look for the "hardest" problems in the above classes, that is, problems which cannot have stronger approximation properties unless $\mathsf{P} = \mathsf{NP}$. In complexity theory, saying that a problem is the hardest one in a class is equivalent to saying that it is complete for that class with respect to a suitable reducibility. The goal of the next section is to identify the right notion of such reducibility.

3. Which reducibility?

It should be clear that the many-to-one polynomial-time reducibility is inadequate to study the approximability properties of optimization problems. Indeed, if we want to map an optimization problem A into an optimization problem B then we need not only a polynomial-time computable function f mapping instances of A into instances of B but also a polynomial-time computable function g mapping back solutions of B into solutions of A (see Fig. 1).

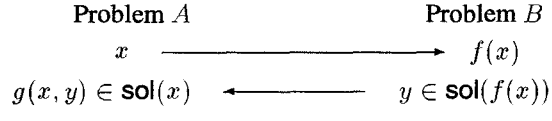

Figure 1. The general reduction scheme

Consistent with the general scheme shown in the above figure, several approximation preserving reducibilities have been defined in the literature differing for the way they preserve the performance ratios and/or for the additional information used by functions f and g. Before proceeding to describe these different reducibilities, we note that the *metric reducibility* defined in [30] does not really fit into this framework. Indeed, this reducibility allows one to compute function opt_A by using an algorithm to compute opt_B without preserving any level of approximability but the best one (that is, performance ratio equal to 1). It is thus not surprising that the class of complete problems introduced in [30] includes problems with drastically different approximability properties.

More adequate for studying approximability properties of optimization problems is the ratio preserving reducibility [37]. This reducibility has been used to show that a variety of NP-hard problems can be interreduced via reductions that preserve the quality of the approximation. However, this reducibility is "non-constructive", that is, it does not include the function g in order to compute an approximate solution. That is why we do not include the ratio preserving reducibility within our taxonomy.

For different reasons we do not include in our review the structure preserving reducibility [6]. This reducibilty is

based on a very detailed examination of the combinatorial structures both of the problems being reduced and of the reductions. In our opinion the conditions under which structure preserving interreducible optimization problems have similar approximability properties are too restrictive to still be of general interest. Nevertheless, it is worth pointing out that by means of the structure preserving reducibility, as far as we know, the first completeness result for optimization problems has been obtained: indeed, in [7] it is implicitly shown that a weighted version of the maximum satisfiability problem is NPO-complete with respect to the structure preserving reducibility.

In the following, we say that a pair (f, g) of polynomial-time computable functions (as in Fig. 1) is a *reduction from* A *to* B. Moreover, we denote by $E(x, y)$ the *absolute error of* y *with respect to* x, that is, $E(x, y) = |\mathsf{opt}(x) - \mathsf{m}(x, y)|$. Finally, we say that a reducibility $\leq$ *preserves membership in a class* $\mathcal{C}$ if $A \leq B$ and $B \in \mathcal{C}$ imply $A \in \mathcal{C}$ and that a reducibility is *approximation preserving* if it preserves membership in either APX, PTAS, or both.

3.1. Strict, A–, and P–reducibility [34, 16]

A reduction (f, g) is said to be a *strict reduction* if, for any $x \in I_A$ and for any $y \in \mathsf{sol}_B(f(x))$, the following holds:

$$R_A(x, g(x, y)) \leq R_B(f(x), y).$$

A reduction (f, g) is said to be an *A-reduction* if a computable function $c : Q \cap (1, \infty) \to Q \cap (1, \infty)$ exists such that, for any $x \in I_A$, for any $y \in \mathsf{sol}_B(f(x))$, and for any $r > 1$, the following holds:

$$R_B(f(x), y) \leq r \Rightarrow R_A(x, g(x, y)) \leq c(r).$$

A reduction (f, g) is said to be a *P-reduction* if a computable function $c : Q \cap (1, \infty) \to Q \cap (1, \infty)$ exists such that, for any $x \in I_A$, for any $y \in \mathsf{sol}_B(f(x))$, and for any $r > 1$, the following holds:

$$R_B(f(x), y) \leq c(r) \Rightarrow R_A(x, g(x, y)) \leq r.$$

Observe that, in [34], an A-reduction (respectively, P-reduction) is called a bounded (respectively, continuous) reduction.

Proposition 6 ([34]) *The A-reducibility (respectively, P-reducibility) preserves membership in* APX *(respectively,* PTAS*).*

PROOF: Let T_B be an r-approximation algorithm for B and let (f, g, c) be an A-reduction from A to B. Then

$$T_A(x) = g(x, T_B(f(x))),$$

is a $c(r)$-approximation algorithm for A. A similar proof holds for the P-reducibility. □

Clearly, every strict reduction is also an A-reduction and a P-reduction (it suffices to set c equal to the identity function). Observe also that a P-reduction is an A-reduction if and only if the inverse of function c is a computable function from $Q \cap (1, \infty)$ to $Q \cap (1, \infty)$.

3.2. Continuous reducibility [41]

A reduction (f, g) is said to be a *continuous reduction* if a positive constant α exists such that, for any $x \in I_A$ and for any $y \in \mathsf{sol}_B(f(x))$,

$$R_A(x, g(x, y)) \leq \alpha R_B(f(x), y).$$

Clearly, a continuous reduction is also an A-reduction (it suffices to define $c(r) = \alpha r$). Thus, the continuous reducibility preserves membership in APX.

3.3. L–reducibility [36]

A reduction (f, g) is said to be an *L-reduction* if two positive constants α and β exist such that:

1. For any $x \in I_A$, $\mathsf{opt}_B(f(x)) \leq \alpha \mathsf{opt}_A(x)$.
2. For any $x \in I_A$ and for any $y \in \mathsf{sol}_B(f(x))$,

$$E_A(x, g(x, y)) \leq \beta E_B(f(x), y).$$

The L-reducibility preserves membership in PTAS. Indeed, this is a consequence of the following result.

Proposition 7 *If a minimization (respectively, maximization) problem* A *reduces to an* NPO *problem* B *via an L-reduction* (f, g, α, β)*, then* A *reduces to* B *via the P-reduction* (f, g, c) *where* $c(r) = 1 + (r - 1)/(\alpha\beta)$ *(respectively,* $c(r) = 1 + (r + 1)/(\alpha\beta r)$*).*

PROOF: Assume that A is a minimization problem. Observe that if (f, g, α, β) is an L-reduction from A to B, then, for any $x \in I_A$ and for any $y \in \mathsf{sol}_B(f(x))$,

$$\frac{E_A(x, g(x, y))}{\mathsf{opt}_A(x)} \leq \alpha\beta \frac{E_B(f(x), y)}{\mathsf{opt}_B(f(x))}.$$

It is also easy to see that

$$\frac{E_B(f(x), y)}{\mathsf{opt}_B(f(x))} \leq R_B(f(x), y) - 1.$$

Thus, we have that

$$R_A(x, g(x, y)) \leq 1 + \alpha\beta \left(R_B(f(x), y) - 1\right).$$

By defining $c(r) = 1 + (r-1)/(\alpha\beta)$, it follows that $R_B(f(x),y) \leq c(r)$ implies $R_A(x,g(x,y)) \leq r$.

A similar proof holds in the case A is a maximization problem. □

From the proof of the above proposition, it also follows that if a minimization problem A is L-reducible to an NPO problem B then A is A-reducible to B. In other words, L-reductions from minimization problems to optimization problems preserve membership in APX. However, there is strong evidence that, in general, this is not true whenever the starting problem is a maximization one [15]. The intuitive reason of this fact is that, when α and β are sufficiently large, the function c in the above proposition is not invertible.

3.4. S–reducibility [13]

A reduction (f,g) is said to be an *S-reduction* if:

1. For any $x \in I_A$, $\mathsf{opt}_B(f(x)) = \mathsf{opt}_A(x)$.
2. For any $x \in I_A$ and for any $y \in \mathsf{sol}_B(f(x))$,

$$\mathsf{m}_A(x,g(x,y)) = \mathsf{m}_B(f(x),y).$$

Clearly, every S-reduction is also a strict reduction, a continuous reduction and an L-reduction.

3.5. E–reducibility [29]

A reduction (f,g) is said to be an *E-reduction* if a polynomial p and a positive constant β exist such that:

1. For any $x \in I_A$, $\mathsf{opt}_B(f(x)) \leq p(|x|)\,\mathsf{opt}_A(x)$.
2. For any $x \in I_A$ and for any $y \in \mathsf{sol}_B(f(x))$,

$$R_A(x,g(x,y)) \leq 1 + \beta\,(R_B(f(x),y) - 1)\,.$$

Clearly, every E-reduction is also an A-reduction and a P-reduction. Indeed, it suffices to define $c(r) = 1+\beta(r-1)$ in the case of the A-reducibility and $c(r) = 1 + (r-1)/\beta$ in the case of the P-reducibility. The E-reducibility is thus the first significant example of a reducibility that preserves membership both in APX and in PTAS.

Observe that, for any function r, an E-reduction maps $r(n)$-approximate solutions (that is, solutions y of an instance x whose performance ratio $R(x,y)$ is bounded by $r(|x|)$) into $(1+\alpha(r(n^h)-1))$-approximate solutions where h is a constant depending only on the reduction. Hence, the E-reducibility not only preserves membership in APX and in PTAS but also membership in poly-APX, that is, the class of problems approximable within a polynomial factor, and log-APX, that is, the class of problems approximable within a logarithmic factor (for a formal definition of these two classes see [29]). It is also worth observing that within the class APX the E-reducibility is just a generalization of the L-reducibility [15].

3.6. PTAS–reducibility [18]

The PTAS-reducibility is the first example of reducibility that allows the functions f and g to depend on the performance ratio. This implies that different constraints have to be put on the computation time of f and g.

A reduction (f,g) is said to be a PTAS-*reduction* if a computable function $c : Q \cap (1,\infty) \to Q \cap (1,\infty)$ exists such that:

1. For any $x \in I_A$ and for any $r > 1$, $f(x,r) \in I_B$ is computable in time $t_f(|x|,r)$.
2. For any $x \in I_A$, for any $r > 1$, and for any $y \in \mathsf{sol}_B(f(x,r))$, $g(x,y,r) \in \mathsf{sol}_A(x)$ is computable in time $t_g(|x|,|y|,r)$.
3. For any fixed r, both $t_f(\cdot,r)$ and $t_g(\cdot,\cdot,r)$ are bounded by a polynomial.
4. For any $x \in I_A$, for any $r > 1$, and for any $y \in \mathsf{sol}_B(f(x,r))$,

$$R_B(f(x,r),y) \leq c(r) \Rightarrow R_A(x,g(x,y,r)) \leq r.$$

Observe that according to this definition, functions like $2^{1/(r-1)}n^h$ or $n^{1/(r-1)}$ are admissible bounds on the computation time of f and g, while this is not true for functions like 2^n.

It is easy to see that the PTAS-reducibility preserves membership in PTAS (this is the motivation for imposing the bounds on the computation time of f and g).

The PTAS-reducibility is a generalization of the P-reducibility. Indeed, the only difference between the PTAS-reducibility and the P-reducibility is the fact that f and g may depend on r: this generalization is necessary to prove completeness results. Indeed, by making use of the PTAS-reducibility it is possible to prove the APX-completeness of MAX SAT and thus of other important optimization problems [18, 29]. On the other hand, it is possible to prove that the APX-completeness of MAX SAT with respect to either the P-reducibility or the L-reducibility would imply that $\mathsf{P}^{\mathrm{SAT}} = \mathsf{P}^{\mathrm{SAT}[\log n]}$ where $\mathsf{P}^{\mathrm{SAT}}$ (respectively, $\mathsf{P}^{\mathrm{SAT}[\log n]}$) denotes the class of languages decidable in polynomial time asking a polynomial (respectively, logarithmic) number of queries to an oracle for the satisfiability problem. This latter event seems to be unlikely and, in any case, the relationship between these two classes is a well-known open question in complexity theory [23].

Red.	Ref.	Additional parameters	Constraints to be satisfied	Membership preserved
$\leq_{\text{strict}}$	[34]		$R_A(x, g(x, y)) \leq R_B(f(x), y)$	all
$\leq_{\text{A}}$	[34]	function c	$R_B(f(x), y) \leq r \Rightarrow R_A(x, g(x, y)) \leq c(r)$	APX
$\leq_{\text{P}}$	[34]	function c	$R_B(f(x), y) \leq c(r) \Rightarrow R_A(x, g(x, y)) \leq r$	PTAS
$\leq_{\text{C}}$	[41]	constant α	$R_A(x, g(x, y)) \leq \alpha R_B(f(x), y)$	APX
$\leq_{\text{L}}$	[36]	constants α, β	$\mathsf{opt}_B(f(x)) \leq \alpha \mathsf{opt}_A(x)$ $E_A(x, g(x, y)) \leq \beta E_B(f(x), y)$	PTAS APX if $\text{type}_A = \min$
$\leq_{\text{S}}$	[13]		$\mathsf{opt}_B(f(x)) = \mathsf{opt}_A(x)$ $\mathsf{m}_A(x, g(x, y)) = \mathsf{m}_B(f(x), y)$	all
$\leq_{\text{E}}$	[29]	polynomial p constant β	$\mathsf{opt}_B(f(x)) \leq p(\lvert x \rvert)\, \mathsf{opt}_A(x)$ $R_A(x, g(x, y)) \leq 1 + \beta\,(R_B(f(x), y) - 1)$	all
$\leq_{\text{PTAS}}$	[18]	ratio r	$R_B(f(x, r), y) \leq c(r) \Rightarrow R_A(x, g(x, y, r)) \leq r$	PTAS
$\leq_{\text{AP}}$	[15]	constant α	$R_B(f(x, r), y) \leq r \Rightarrow R_A(x, g(x, y, r)) \leq 1 + \alpha(r - 1)$	all

Table 1. Summary of the nine reducibilities

3.7. AP–reducibility [15]

A PTAS-reduction (f, g, c) is said to be an *AP-reduction* if a constant α exists such that $c(r) = 1 + (r - 1)/\alpha$. In other words, the fourth condition of the PTAS-reducibility is changed into

$$R_B(f(x, r), y) \leq r \Rightarrow R_A(x, g(x, y, r)) \leq 1 + \alpha(r - 1).$$

Since the AP-reducibility is a restriction of the PTAS-reducibility, it preserves membership in PTAS. Moreover, from the fact that function c is invertible it follows that it also preserves membership in APX. On the other hand, it is clear that the AP-reducibility is a generalization of the E-reducibility. In order to maintain the nice property of this latter reducibility of preserving membership in poly-APX and log-APX, we impose the following further restriction on the computation time of f and g: for any fixed n, both $t_f(n, \cdot)$ and $t_g(n, n, \cdot)$ are non-increasing functions.

3.8. The taxonomy

In Fig. 2 and in Table 1 we summarize all the approximation preserving reducibilities that we have reviewed. Both in the figure and in the table we have denoted by $\leq_{\text{C}}$ (respectively, $\leq_{\text{strict}}$, $\leq_{\text{A}}$, $\leq_{\text{P}}$, $\leq_{\text{L}}$, $\leq_{\text{S}}$, $\leq_{\text{E}}$, $\leq_{\text{PTAS}}$, and $\leq_{\text{AP}}$) the continuous (respectively, strict, A-, P-, L-, S-, E-, PTAS-, AP-) reducibility.

We do not claim that this taxonomy covers *all* the approximation preserving reducibilities that have appeared in the literature. However, we are very confident that the reducibilities we missed are restrictions of one of those included in the taxonomy.

Observe that the fact that the taxonomy does not con-

verge into one reducibility is due to the fact that we did not require that the inverse of function c in the definitions of the two reducibilities is computable. If we do so, then any A-reduction turns out to be a PTAS-reduction, while the opposite not necessarily holds. Since, as far as we know, no approximation preserving reduction that has appeared in the literature makes use of a non-invertible function c, we recommend the PTAS-reducibility as the first effective candidate to be used. This choice allows one to rephrase all approximation preserving reduction results that have been obtained within class APX in terms of PTAS-reducibility.

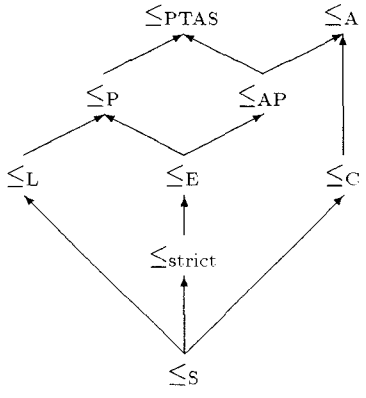

Figure 2. The taxonomy of approximation preserving reducibilities

In order to deal with classes larger than APX (such as log-APX, poly-APX, and NPO), the AP-reducibility seems, instead, to be more adequate since, as we have already observed, this reducibility preserves membership in all the above classes. In other words, when dealing with problems in one of these classes we recommend the AP-reducibility as the second effective candidate to be used.

Finally, a distinction has to be made depending on the type of results one wants to obtain. In spite of the fact that the L-reducibility cannot be used to obtain completeness results in classes such as APX, log-APX, and poly-APX [15], as a matter of fact it seems to be the most widely used when the goal is not a completeness result but a non-approximability result. This is mainly due to its simple and natural definition: somehow, such a simple definition already suggests what to do and how to obtain it. It is for this reason that we recommend the L-reducibility as the third effective candidate to be used.

In Table 2 we summarize the above discussion showing for each class and for each type of result the most effective candidate to be used. This table also suggests to first try to use the L-reducibility in order to obtain a non-approximability result. Successively, if completeness is one's goal, then the L-reduction must be turned into either an AP-reduction or a PTAS-reduction.

classes	inapproximability	completeness
NPO	$\leq_L$	$\leq_{AP}$
poly-APX		
log-APX		
APX		$\leq_{PTAS}$

Table 2. The three candidates

As we will see in the next section, sometimes it is useful to switch from the L-reducibility to one of the other two reducibilities just because a more powerful reducibility may allow one to obtain simpler reductions. Simple reductions, in turn, can be easily explained, checked, and interpreted.

4. How to reduce?

Once the reducibility to be used has been decided (that is, the L-reducibility, the AP-reducibility, or the PTAS-reducibility), a harder task still awaits the reader, that is, the task of effectively reducing to the target problem A another problem B for which non-approximability results are already known.

4.1. Using NP–hardness proofs

To this aim, the first thing one should do is to check the NP-hardness proof of A (if any exists). It can likely happen that this proof is already an approximation preserving reduction: if the starting problem of the NP-hardness proof has the desired non-approximability properties, then you are done. As an example of this "technique", let us prove that MAX 3SAT is L-reducible to MAX 2SAT by using the 20-year old NP-hardness proof of MAX 2SAT.

Theorem 8 ([21]) MAX 3SAT *L-reduces to* MAX 2SAT.

PROOF: Given an instance φ of MAX 3SAT with m clauses, let $a_i \vee b_i \vee c_i$ be the ith clause, for $i = 1, \ldots, m$, where each a_i, b_i, and c_i represents either a variable or its negation (note that without loss of generality, we may assume that each clause contains exactly three literals). To this clause, we associate the following ten new clauses with at most two literals per clause:

1. a_i
2. b_i
3. c_i

4. d_i
5. $\neg a_i \vee \neg b_i$
6. $\neg a_i \vee \neg c_i$
7. $\neg b_i \vee \neg c_i$
8. $a_i \vee \neg d_i$
9. $b_i \vee \neg d_i$
10. $c_i \vee \neg d_i$

where d_i is a new variable. Let $f(\varphi) = \varphi'$ be the resulting instance of MAX 2SAT.

Observe that, for any truth assignment satisfying the ith clause, a truth setting for d_i exists causing precisely seven of the above ten clauses to be satisfied: indeed, if only one (respectively, two) among a_i, b_i, and c_i is (respectively, are) true, then we set d_i to false (respectively, true), otherwise d_i may be either true or false. Moreover, for any truth assignment for which a_i, b_i, and c_i are false, no truth setting of d_i can cause more than six of the ten clauses to be satisfied.

This implies that

$$\mathsf{opt}(\varphi') = 6m + \mathsf{opt}(\varphi) \leq 13\mathsf{opt}(\varphi)$$

where the last inequality is due to the fact that $\mathsf{opt}(\varphi) \geq m/2$ (see [26]). Finally, for any truth assignment τ' to the variables of φ', its restriction $g(\varphi, \tau') = \tau$ to the variables of φ is such that

$$\begin{aligned} \mathsf{opt}(\varphi) - \mathsf{m}(\varphi, \tau) &= \mathsf{opt}(\varphi') - 6m - \mathsf{m}(\varphi, \tau) \\ &\leq \mathsf{opt}(\varphi') - 6m - \mathsf{m}(\varphi', \tau') + 6m \\ &= \mathsf{opt}(\varphi') - \mathsf{m}(\varphi', \tau'). \end{aligned}$$

It follows that f and g are an L-reduction from MAX 3SAT to MAX 2SAT with $\alpha = 13$ and $\beta = 1$. □

Even if the following statement has not been "empirically" checked, we claim that a large fraction of the NP-hardness proofs that have appeared in the literature turn out to be approximation preserving. The basic idea behind this claim is that the NP-completeness proofs normally use the fact that, after all, most NP-complete problems originate from optimization problems.

4.2. Expander graphs

Even if the original NP-hardness proof is not approximation preserving, it can still be useful. Indeed, in several cases this proof can be transformed into an approximation preserving reduction by means of some "expanding" modifications. Typical examples of this fact are the non-approximability results for the MAXIMUM 3-DIMENSIONAL MATCHING [27] and for the MINIMUM GRAPH COLORING [32]. Intuitively, these modifications try to restrict the solutions of the target problem to a subset of "good" solutions: whenever a solution which is not in this subset is computed, it can be transformed into a good solution in polynomial time and without any loss in the measure.

The first application of this technique made use of expander graphs in order to prove that MAX 3SAT is L-reducible to MAX 3SAT(B), that is, MAX 3SAT restricted to instances in which each variable appears at most B times [36]. In the following, however, we will describe the technique as presented in [2].

Recall that the standard NP-hardness proof is based on the following idea. If a variable y occurs h times, then create h new variables y_i, substitute the ith occurrence of y with y_i, and add the following $h-1$ constraints: $y_1 \equiv y_2$, $y_2 \equiv y_3$, ..., $y_{h-1} \equiv y_h$ (observe that each constraint $y_i \boxminus y_j$ may be expressed by the two clauses $y_i \vee \neg y_j$ and $\neg y_i \vee y_j$). Graphically, the new clauses can be represented as a simple path of h nodes as shown in Fig. 3 in the case of $h = 5$. In the figure, each edge (y_i, y_j) corresponds to the two clauses $y_i \vee \neg y_j$ and $\neg y_i \vee y_j$.

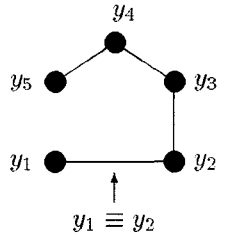

Figure 3. The equivalence path

This reduction is not approximation preserving since deleting just one edge of the graph may allow to assign inconsistent truth-values to the h copies of y and to satisfy an arbitrarily large number of the clauses corresponding to the original ones (for example, consider the case in which the original formula contains $h/2$ copies of the clause containing only y and $h/2$ copies of the clause containing only $\neg y$). The problem is that a simple path admits "almost balanced" partitions of its nodes whose corresponding number of cut-edges is small. The solution to this problem consists of "enforcing" the bonds that link the copies of the same variable. A possible such enforcement could consist of substituting the path of Fig. 3 with a complete graph (see Fig. 4).

Observe that this new graph corresponds to the following set of equivalence constraints:

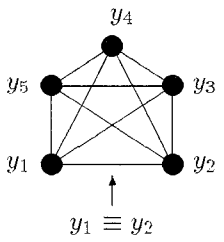

Figure 4. The equivalence complete graph

$$\{y_i \equiv y_j : 1 \leq i < j \leq h\}.$$

In this case, any proper subset T of the nodes of the graph yields a number of cut-edges equal to $|T|(h - |T|)$ which is always greater than or equal to $|T|$. If a truth-assignment does not assign the same value to all copies of y, then these copies are divided into two sets according to the value they have received. Let T be the smaller of these two sets and consider the corresponding set of vertices of the complete graph. At least $|T|$ edges leave this set and each of them corresponds to an unsatisfied equivalence constraint (that is, an unsatisfied clause). Hence, by changing the truth-values of these $|T|$ copies, we gain at least $|T|$ clauses "belonging" to the equivalence complete graph. On the other hand, at most $|T|$ of the clauses corresponding to the original ones can now be unsatisfied. This implies that we do not loose anything if we impose that any truth-assignment assigns the same value to all the copies of one variable.

This approach, however, has two disadvantages. First, the number of occurrences of a variable is no more bounded by a constant smaller than h (actually, we have $2h - 1$ occurrences of each copy). Second, the measure of a solution for the new instance is no more bounded by a linear function of the measure of the corresponding solution for the original instance. Indeed, if a truth-assignment satisfies k clauses in the original formula, then the corresponding truth-assignment for the new formula satisfies $k + \sum_{i=1}^{n}(h_i - 1)h_i$ clauses where n denotes the number of variables and h_i denotes the number of occurrences of the ith variable. This implies that this reduction would not be an L-reduction.

Actually, what we need is a graph which is an expanded version of the path of Fig. 3 (or, equivalently, a sparsified version of the complete graph of Fig. 4) with maximum degree bounded by a constant (this property is satisfied by the path) and such that any "almost balanced" partition of its nodes yields a large number of cut-edges (this property is satisfied by the complete graph). Such graphs exist and are well-known: they are the *expander graphs* (or at least one of their variations).

Definition 9 *A d-regular graph $G = (V, E)$ is an* expander *if, for any $S \subset V$, the number of edges between S and $V - S$ is at least* $\min(|S|, |V - S|)$.

Theorem 10 ([31]) *A polynomial-time algorithm T and a fixed integer N exist such that, for any $n > N$, $T(n)$ produces a 14-regular expander E_n with n nodes.*

To be precise, the algorithm of the above theorem produces a graph with $n(1 + o(1))$ nodes. However, this does not affect the proof of the following result.

Theorem 11 ([36, 2]) MAX 3SAT *is L-reducible to* MAX 3SAT(29).

PROOF: Let φ be an instance of MAX 3SAT with m clauses. For each variable y, let h_y be the number of occurrences of y (without loss of generality, we may assume that $h_y \geq N$ where N is the fixed integer of Theorem 10).

Let E_{h_y} be a 14-regular expander. For each node u of E_{h_y}, we create one new variable y_u and, for each edge (u, v) of E_{h_y}, we create two new clauses $y_u \vee \neg y_v$ and $\neg y_u \vee y_v$. Finally, we substitute the ith occurrence of y in the original clauses with y_u where u is the ith node of E_{h_y}. This concludes the definition of function f. Observe that each variable occurs at most 29 times (28 times in the clauses corresponding to edges of the expander and one time in the original clause): that is, $\varphi' = f(\varphi)$ is indeed an instance of MAX 3SAT(29).

From the above discussion, it follows that we can now restrict ourselves to truth-assignments for the variables of φ' that assign the same value to all copies of the same variable. Given such an assignment τ', we define a truth-assignment $\tau = g(\varphi, \tau')$ for the variables of φ by assigning to y the value of its copies.

We then have that

$$\mathsf{opt}(\varphi') = 14\sum_{y} h_y + \mathsf{opt}(\varphi) \leq 85\mathsf{opt}(\varphi)$$

where the last inequality is due to the fact that each clause contains at most three literals and to the fact that $\mathsf{opt}(\varphi) \geq m/2$ [26].

Moreover, we have that, for any truth-assignment τ' to the variables of φ', the corresponding truth-assignment τ to the variables of φ satisfies the following equality:

$$\mathsf{opt}(\varphi) - \mathsf{m}(\varphi, \tau) = \mathsf{opt}(\varphi') - \mathsf{m}(\varphi', \tau').$$

That is, (f, g) is an L-reduction with $\alpha = 85$ and $\beta = 1$ from MAX 3SAT to MAX 3SAT(29). □

Observe that reducing MAX 3SAT(29) to MAX 3SAT(5) is an easier task: indeed, in this case it is sufficient to use graphs similar to that of Fig. 3.

After [36], other variations of expander graphs have been used to obtain approximation preserving reductions [1, 12, 17]. In some cases, this technique has also led to the definition and the proof of existence of types of expanders that were not previously known [17].

4.3. Randomized perturbing

The second technique that we describe is based on the following rule of thumb (widely accepted in the computer science community): whenever you do not know what to do, flip a coin. In a few words, this technique makes use of randomization to "perturb" a feasible solution of the target problem in order to obtain a feasible solution of the original problem (that is, randomization is used to compute function g). For this reason, we call this technique *randomized perturbing*. Before describing the technique more precisely, we recall that randomization has been used for developing approximation algorithms [40] and has also been used to compute the function f of an approximation preserving reduction [17]. As far as we know, the randomized perturbing technique is the first example of application of the probabilistic method to the computation of function g.

We will describe the technique by referring to MAX SAT and MAX 3SAT. In particular, we will show how a 2-approximation algorithm for MAX 3SAT can be used to develop a 3-approximation algorithm for MAX SAT. This may seem somehow useless since a 2-approximation algorithm for MAX SAT is known since 1974 [26]. However, we have chosen this example because it is simple enough to be described in one page but, at the same time, sufficiently explicative of the approach.

Recall that standard reduction techniques based on local replacement [20] fail to approximation preserving reduce MAX SAT to MAX 3SAT due to the possible presence of large clauses. Large clauses, however, are easy to satisfy using a simple randomized algorithm (that, in turn, performs poorly on small clauses) [26]. We then *combine* (in a probabilistic sense) a solution based on standard reductions and one given by the randomized algorithm, and this mixed solution will be good for any combination of small and large clauses. The idea of probabilistically combining different solutions has been already used in the design of approximation algorithms (e.g. [22]).

Theorem 12 ([19]) *If a 2-approximation algorithm exists for* MAX 3SAT, *then* MAX SAT *admits a 3-approximation algorithm.*

PROOF: Let φ be an instance of MAX SAT with m clauses, m_3 be the number of clauses of φ with 3 or less literals, and φ_3 be instance φ restricted to these m_3 clauses. Define $m_l = m - m_3$ and $\varphi_l = \varphi - \varphi_3$. Let τ_3 be a 2-approximate solution for φ_3 (which, by hypothesis, is computable in polynomial time), and a be the number of clauses satisfied by τ_3. It is immediate to verify that

$$\mathsf{opt}(\varphi) \leq 2a + m_l \,. \tag{1}$$

Indeed, any assignment cannot satisfy more than $2a$ clauses in φ_3 (otherwise τ_3 would not be 2-approximate) and more then m_l clauses in φ_l. We now define a random assignment τ_R over the variables of φ with the following distribution:

- If a variable x occurs in φ_3, then

$$\mathbf{Pr}[\tau_R(x) = \tau_3(x)] = \frac{3}{4} \,.$$

- If a variable x occurs in φ but not in φ_3, then

$$\mathbf{Pr}[\tau_R(x) = \mathsf{true}] = \frac{1}{2} \,.$$

Let us now estimate the average number of clauses of φ that are satisfied by τ_R. Since any literal is true with probability at least 1/4, the probability that a clause in φ_l is not satisfied is at most $(3/4)^4$. On the other hand, if a clause is satisfied by τ_3, then there is a probability at least $3/4$ that it is still satisfied by τ_R. We can thus infer a lower bound on the average number of clause of φ that are satisfied by τ_R:

$$\mathbf{E}[\mathsf{m}(\varphi, \tau_R)] \geq \frac{3}{4}a + \left(1 - \left(\frac{3}{4}\right)^4\right) m_l \geq \frac{1}{3}\mathsf{opt}(\varphi)$$

where the last inequality is due to (1). Using the method of conditional probabilities [33], we can find in linear time an assignment τ such that $\mathsf{m}(\varphi, \tau) \geq \mathbf{E}[\mathsf{m}(\varphi, \tau_R)]$. That is, the performance ratio of τ is at most 3. □

The previous theorem can be easily generalized in order to obtain the following result.

Theorem 13 ([19]) MAX SAT *is* PTAS*-reducible to* MAX 3SAT.

We point out that in the proof of the above theorem, the authors make use of an "intermediate" problem in order to go from MAX SAT to MAX 3SAT. This problem is not fixed but depends on the approximation factor that we want to preserve. This is not in contradiction with the definition of PTAS-reduction. Indeed, the only known proof of the analogue of the above result with respect to the L-reducibility is based on the PCP theorem [3] and thus makes use of much more complicated techniques [29].

It is also worth observing that the previous result is an example of the fact that approximation preserving reductions are not only a method of proving non-approximability results, but they are very useful for finding positive results

as well (even if one has to be a bit careful when using the L-reducibility for this). Indeed, in [19] it is shown that the reduction from MAX SAT to MAX 3SAT can be used to obtain a more efficient approximation scheme for the "planar" version of MAX SAT [25, 28].

5. Conclusions

> MOI: Stojil, Stojil, tu m'avais pourtant juré que tu étais immortel!
>
> LUI: C'est vrai, mais je ne t'ai jamais juré que j'étais infaillible.
>
> Daniel Pennac [39]

The first goal of this paper has been to guide the reader through the wide range of definitions of approximation preserving reducibilities that can be used for proving non-approximability results. At the end of this guided tour, we have recommended three reducibilities as the most effective candidates to be used: the L-reducibility, the AP-reducibility, and the PTAS-reducibility. While the first one was seen to be natural, simple, and, in most cases, sufficient, the other two reducibilities were seen to be more powerful, which makes it possible to prove more results or to simplify reductions that would otherwise be overly complicated. Nevertheless, we do not think that the last word about approximation preserving reducibilities has been said: it is still possible that new reducibilities have their role to play. Even if we do not encourage the reader to define new reducibilities when it is not necessary, we think that reconsidering this possibility is an ultimate alternative when all other attempts seem to fail. Coming back to our initial analogy with the process of buying a computer, after all, defining new reducibilities is much less expensive than producing new computers!

The second goal has been to suggest some guidelines to be followed when trying to develop an approximation preserving reduction. To this aim, we have emphasized two techniques that we believe will play an ever-increasing important role in this field: the use of expander graphs and the use of randomization. Both techniques are well-known in the theoretical computer science community and have been used mainly to design new algorithms: only recently they have been used in the field of approximation preserving reductions. This also suggests that other consolidated techniques in other fields (mainly, those related to algorithm engineering) may play an important role in order to obtain new or better non-approximability results. Support for this claim comes from recent applications of error-correcting codes [42] and of linear programming [43]. In our opinion, indeed, this is one of the main novelties of approximation preserving reductions with respect to NP-hardness proofs.

We want to conclude by recalling that, whenever no problem has been found that is reducible to the target problem A, there is still another possibility: the non-approximability result for A may be derived directly from the PCP theorem (or from one of its variations [9]) by making use, for instance, of the gadget method introduced in [9] (this method has the advantage that the gadgets can be constructed automatically [43]). This alternative may be summarized in the following statement (which is a slight modification of a motto that has been used in the 80s): *real men reduce from* PCP!

Acknowledgments Before writing this paper, I have benefited from conversations with Christos Papadimitriou and Luca Trevisan. I would like to express my gratitude to Viggo Kann, Alessandro Panconesi, Riccardo Silvestri, and Luca Trevisan for pointing out many corrections which have improved the quality of the manuscript. Finally, I am grateful to Larry Winter for providing numerous helpful suggestions that have enforced a consistency of grammatical style that appears to be foreign to my natural way of writing.

References

[1] E. Amaldi and V. Kann. The complexity and approximability of finding maximum feasible subsystems of linear relations. *Theoretical Computer Science*, 147:181–210, 1995.

[2] S. Arora and C. Lund. *Hardness of approximations*, chapter 10 of [24]. PWS, 1997.

[3] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *Proceedings of the 33rd Symposium on Foundations of Computer Science*, pages 14–23. IEEE, 1992.

[4] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti Spaccamela, and M. Protasi. *Approximate solution of NP-hard optimization problems*. Springer Verlag, 1997.

[5] G. Ausiello, P. Crescenzi, and M. Protasi. Approximate solution of NP optimization problems. *Theoretical Computer Science*, 150:1–55, 1995.

[6] G. Ausiello, A. D'Atri, and M. Protasi. Structure preserving reductions among convex optimization problems. *Journal of Computer and System Sciences*, 21:136–153, 1980.

[7] G. Ausiello, A. D'Atri, and M. Protasi. Lattice theoretic ordering properties for NP-complete optimization problems. *Annales Societatis Mathematicae Polonae*, 4:83–94, 1981.

[8] J. Balcázar, J. Díaz, and J. Gabarró. *Structural complexity I*. Springer-Verlag, 1988.

[9] M. Bellare, O. Goldreich, and M. Sudan. Free bits, PCPs and non-approximability – towards tight results. In *Proceedings of the 36th Symposium on Foundations of Computer Science*, pages 422–431. IEEE, 1995.

[10] D. Bovet and P. Crescenzi. *Introduction to the Theory of Complexity*. Prentice Hall, 1993.

[11] D. Bruschi, D. Joseph, and P. Young. A structural overview of NP optimization problems. *Algorithms Review*, 2:1–26, 1991.

[12] A. Clementi and L. Trevisan. Improved non-approximability results for vertex cover problems with density constraints. In *Proc. 2nd Conference on Computing and Combinatorics*, number 1090 in Lecture Notes in Computer Science, pages 333–342. Springer Verlag, 1996.

[13] P. Crescenzi, C. Fiorini, and R. Silvestri. A note on the approximation of the MAX CLIQUE problem. *Information Processing Letters*, 40:1–5, 1991.

[14] P. Crescenzi and V. Kann. A compendium of NP optimization problems. Technical Report SI/RR - 95/01, Dipartimento di Scienze dell'Informazione, University of Rome, 1995. The list is updated continuously. The latest version is available as http://www.nada.kth.se/theory/problemlist.html.

[15] P. Crescenzi, V. Kann, R. Silvestri, and L. Trevisan. Structure in approximation classes. In *Proc. 1st Conference on Computing and Combinatorics*, number 959 in Lecture Notes in Computer Science, pages 539–548. Springer Verlag, 1995.

[16] P. Crescenzi and A. Panconesi. Completeness in approximation classes. *Information and Computation*, 93:241–262, 1991.

[17] P. Crescenzi, R. Silvestri, and L. Trevisan. To weight or not to weight: Where is the question? In *Proc. 6th Israeli Symposium on Theory of Computing Systems*, pages 68–77. IEEE, 1996.

[18] P. Crescenzi and L. Trevisan. On approximation scheme preserving reducibility and its applications. In *Proc. 14th Conference on Foundations of Software Technology and Theoretical Computer Science*, number 880 in Lecture Notes in Computer Science, pages 330–341. Springer Verlag, 1994.

[19] P. Crescenzi and L. Trevisan. MAXNP-completeness made easy. Technical Report SI/RR - 97/01, DSI, University of Rome, 1997.

[20] M. Garey and D. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness*. Freeman, 1979.

[21] M. Garey, D. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[22] M. Goemans and D. Williamson. New 3/4-approximation algorithms for the maximum satisfiability problem. *SIAM Journal on Discrete Mathematics*, 7:656–666, 1994.

[23] L. Hemaspaandra. Complexity theory column 5: the not-ready-for-prime-time conjectures. *SIGACT News*, 1994.

[24] D. Hochbaum, editor. *Approximation algorithms for NP-hard problems*. PWS, 1997.

[25] H. B. Hunt III, M. V. Marathe, V. Radhakrishnan, S. S. Ravi, D. J. Rosenkrantz, and R. E. Stearns. Approximation schemes using L-reductions. In *Proc. 14th Conference on Foundations of Software Technology and Theoretical Computer Science*, number 880 in Lecture Notes in Computer Science, pages 342–353. Springer Verlag, 1994.

[26] D. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[27] V. Kann. Maximum bounded 3-dimensional matching is MAX SNP-complete. *Information Processing Letters*, 37:27–35, 1991.

[28] S. Khanna and R. Motwani. Towards a syntactic characterization of PTAS. In *Proceedings of the 28th Symposium on Theory of Computing*. ACM, 1996.

[29] S. Khanna, R. Motwani, M. Sudan, and U. Vazirani. On syntactic versus computational views of approximability. In *Proceedings of the 35th Symposium on Foundations of Computer Science*, pages 819–830. IEEE, 1994.

[30] M. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36:490–509, 1988.

[31] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8:261–277, 1988.

[32] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *Journal of ACM*, 41:960–981, 1994.

[33] R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge University Press, 1995.

[34] P. Orponen and H. Mannila. On approximation preserving reductions: Complete problems and robust measures. Technical Report C-1987-28, Department of Computer Science, University of Helsinki, 1987.

[35] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1993.

[36] C. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

[37] A. Paz and S. Moran. Non deterministic polynomial optimization problems and their approximations. *Theoretical Computer Science*, 15:251–277, 1981.

[38] D. Pennac. *La fée carabine*. Gallimard, 1987.

[39] D. Pennac. *Monsieur Malaussène*. Gallimard, 1996.

[40] P. Raghavan. Randomized approximation algorithms in combinatorial optimization. In *Proc. 14th Conference on Foundations of Software Technology and Theoretical Computer Science*, number 880 in Lecture Notes in Computer Science, pages 300–317. Springer Verlag, 1994.

[41] H. Simon. Continous reductions among combinatorial optimization problems. *Acta Informatica*, 26:771–785, 1989.

[42] L. Trevisan. When Hamming meets Euclid: the approximability of geometric TSP and MST. In *Proceedings of the 29th Symposium on Theory of Computing*. ACM, 1997.

[43] L. Trevisan, G. Sorkin, M. Sudan, and D. Williamson. Gadgets, approximation, and linear programming. In *Proceedings of the 37th Symposium on Foundations of Computer Science*, pages 617–626. IEEE, 1996.