Computer assisted proof of optimal approximability results

Uri Zwick *

Abstract

We obtain computer assisted proofs of several spherical volume inequalities that appear in the analysis of semidefinite programming based approximation algorithms for Boolean constraint satisfaction problems. These inequalities imply, in particular, that the performance ratio achieved by the MAX 3-SAT approximation algorithm of Karloff and Zwick is indeed $\frac{7}{8}$, as conjectured by them, and that the performance ratio of the MAX 3-CSP algorithm of the author is indeed $\frac{1}{2}$. Other results are also implied. The computer assisted proofs are obtained using a system called $\mathcal{R}$EAL$\mathcal{S}$EARCH, written by the author. This system uses *interval arithmetic* to produce *rigorous proofs* that certain collections of constraints in real variables have *no* real solution.

1 Introduction

Goemans and Williamson [GW95] used semidefinite programming to obtain a 0.87856-approximation algorithm for the MAX CUT problem. Karloff and Zwick [KZ97] used extensions of their ideas to obtain an approximation algorithm for MAX 3-SAT with a *conjectured* performance ratio of $\frac{7}{8}$. The author [Zwi98] used further extensions of these ideas to obtain an approximation algorithm for MAX 3-CSP with a conjectured performance ratio of $\frac{1}{2}$. (An instance of MAX 3-CSP is composed of an arbitrary collection of constraints each involving at most three Boolean variables. In a MAX 3-SAT instance, each constraint requires the disjunction of up to three literals to evaluate to true. The goal in both cases is to find a Boolean assignment to the variables that maximizes the number, or weight, of the satisfied constraints.) What makes the conjectured performance ratios for the MAX 3-SAT and MAX 3-CSP problems especially interesting is that they completely match inapproximability results obtained for these problems by Håstad [Hås97]. Håstad's ground breaking results follow an equally impressive sequence of works on Probabilistically Checkable Proofs (PCP's) by Feige *et al.* [FGL$^+$96], Arora and Safra [AS98], Arora *et al.* [ALM$^+$98], Bellare *et al.* [BGS98], Raz [Raz98], and others. It follows from Håstad's results that, for any $\epsilon > 0$, obtaining a $\frac{7}{8} + \epsilon$ ratio for MAX 3-SAT and a $\frac{1}{2} + \epsilon$ ratio for MAX 3-CSP are both NP-hard tasks. In [KZ97] and [Zwi98] it was shown that the validity of the $\frac{7}{8}$ and $\frac{1}{2}$ conjectures would follow if certain inequalities involving spherical volumes could be shown to hold. However, no proof of these inequalities was presented. In [KZ97], a mostly analytic proof of one of these inequalities was given. This was sufficient to show that the performance ratio of the MAX 3-SAT algorithm on *satisfiable* instances is $\frac{7}{8}$. Here we use much more sophisticated techniques to obtain computer assisted proofs of *all* the inequalities required to prove the $\frac{7}{8}$ and $\frac{1}{2}$ conjectures, and some other claims.

We believe that the results contained in this paper are important for the following reasons. First, they provide the first rigorous proofs of the results claimed (or conjectured) in [KZ97] and [Zwi98]. The techniques used to obtain these proofs are *not* simple extensions of ideas used in [KZ97] and [Zwi98]. Although the obtained proofs are computer assisted, some nontrivial analytical arguments are needed to transform the claims to be proved into claims that could be verified automatically. Further complications result from the non-elementary nature of the spherical volume function. Second, we hope that the methodology used to obtain the computer assisted proofs, and the software tool developed, the $\mathcal{R}$EAL$\mathcal{S}$EARCH system, would be useful in other situations encountered in the analysis of algorithms. Although the inequalities considered in this paper all involve spherical volumes, similar techniques could be used in other situations. Third, the ideas used in the design of $\mathcal{R}$EAL$\mathcal{S}$EARCH have some interesting theoretical consequences that may be of interest in their own right.

There are by now some well known examples of computer assisted proofs. Perhaps the most famous example of this kind is the proof of the *four color theorem* by Appel and Haken [AH77, AHK77]. (A simpler, more recent, proof by Robertson *et al.* [RSST97] is also computer-assisted.) The four color problem is a *discrete* problem. Even more relevant to our results are, the recent proof by Hales [Hal97a, Hal97b, Hal98, Hal00] of Kepler's conjecture, his proof of the honeycomb conjecture [Hal01], the proof by Hales and McLaughlin [HM98] of the dodecahedral conjecture, the proof by Hass and Schlafly [HS00] of the double bubble conjecture, and a recent proof by Gabai *et al.* [GMT00] of a claim regarding hyperbolic 3-manifolds. All these claims are *continuous* in nature and their computer-assisted proofs,

*School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. E-mail address: zwick@tau.ac.il. This research as supported by THE ISRAEL SCIENCE FOUNDATION (grant no. 246/01).

like our proofs, use *interval arithmetic* in an essential way.

Interval arithmetic (see, e.g., Hansen [Han92]) is a *rigorous* method of carrying out computations involving real numbers with an automatic monitoring of *all* the numerical errors involved. The result of a computation carried out using interval arithmetic is not a single number, but rather an interval that is *guaranteed* to contain the correct answer, in spite of all the numerical errors that may occur in the computation that is carried out using the inherently inexact *floating point* arithmetic. The floating point arithmetic used should, of course, satisfy some basic requirements. In his proof of Kepler's conjecture, Hales [Hal98] relies, for example, on the strict conformity of the processor that executes his 'proof' to the IEEE-754 floating point standard [ANS85] (see also Goldberg [Gol91], Kahan [Kah96b],[Kah96a] and Overton [Ove01]). Almost all the commercially available processors do conform to the IEEE-754 standard. Our proofs rely on even weaker assumptions. Further discussion of this point and of interval arithmetic, floating point arithmetic and the exact assumptions that we make is deferred to Sections 5 and 6 and to the full version of the paper. [1] The broader question of how computer assisted proofs should be viewed by mathematicians and by computer scientists is briefly discussed in Section 7.

The rest of the paper is organized as follows. In the next section we discuss the relation of the results reported here to previously available results. In Section 3 we present the spherical volume inequalities that have to be proved to establish the results of [KZ97] and [Zwi98]. In Section 4 we sketch the partially computer assisted proof of the most difficult of these inequalities. We show there, in particular, how to reduce the task of proving the inequality into the task of proving a claim that can be verified automatically using $\mathcal{R}$EAL$\mathcal{S}$EARCH. (The reader may want to skip Section 4 at first reading.) In Section 5 we very briefly sketch the simple, yet powerful, ideas on which interval arithmetic is based. In Section 6 we briefly describe the $\mathcal{R}$EAL$\mathcal{S}$EARCH system. ($\mathcal{R}$EAL$\mathcal{S}$EARCH is written in C++. The complete source code of $\mathcal{R}$EAL$\mathcal{S}$EARCH can be downloaded from `http://www.cs.tau.ac.il/~zwick/RealSearch.html`. The reader is encouraged to download the code and read it carefully, as it forms part of the proof.) We end in Section 7 with some concluding remarks and a general discussion of computer assisted proofs.

[1] We note in passing that it is not too difficult to implement the basic floating point operations in software, using only integer arithmetic, in a way that would conform to the IEEE-754 standard. (See, e.g., Hauser [Hau98].) This would remove the reliance on the hardware implementation of the floating point arithmetic, but would lengthen the code that would have to be checked as part of the proof. None of the authors cited above thought that this was necessary.

2 Our results and their relation to other results

There is by now a large collection of papers, by many authors, that use semidefinite programming to obtain improved approximation algorithms for various combinatorial optimization problems. This section explains the relation of the results obtained here to these previously available papers.

Goemans and Williamson [GW95] were the first to use semidefinite programming to obtain an improved approximation algorithms. They obtained an 0.87856-approximation algorithm for the MAX CUT and MAX 2-SAT problems, and an 0.79607-approximation algorithm for the MAX DI-CUT problem. Goemans and Williamson [GW95] use purely analytical techniques to prove their results.

Feige and Goemans [FG95] presented improved approximation algorithms for the MAX 2-SAT and MAX DI-CUT problems. They claim that the approximation ratios achieved by these algorithms are 0.931 and 0.859, respectively. The analysis of these algorithms turns out to be much more difficult than the analysis of the original algorithms of Goemans and Williamson [GW95]. Some of the complications result from the use of *rotation functions*. The approximation ratios of the new algorithms are the *global* minima of constrained non-linear minimization problems in *three* real variables. Feige and Goemans [FG95] were not able to produce traditional analytical proofs that the global minima of these problems are indeed at least 0.931 and 0.859, respectively. They based their claims on what might be called *numerical evidence*.

Karloff and Zwick [KZ97] used extensions of these techniques to obtain an approximation algorithm with a *conjectured* approximation ratio of $\frac{7}{8}$. (Note the question mark in the title of the paper.) Although the approximation algorithm itself is very simple, and no rotation function is used in the rounding phase, the analysis of this algorithm is much harder than that of all the previous algorithms. The increased difficulty results from the fact that the approximation ratio of the algorithm is now the global minimum of a constrained minimization problem in *six* variables, and not just three as before. Furthermore, the objective function of this minimization problem is defined in terms of the volume function of *spherical tetrahedra*, a fairly complicated, and probably non-elementary, function. Karloff and Zwick [KZ97] give *numerical evidence* that shows that the ratio of their algorithm is $\frac{7}{8}$, but give no rigorous proof. They do give, however, an almost complete proof of a weaker claim that the performance ratio of their algorithm is $\frac{7}{8}$ on *satisfiable* instances. This proof is mostly analytic but does rely at one point on calculations carried out in Mathematica. (See discussion of this point in Section 4.)

Zwick [Zwi98] used further extensions of these techniques to obtain approximation algorithms for many constraint sat-

isfaction problems that involving at most three variables per constraint. Among the results obtained is a $\frac{1}{2}$-approximation algorithm for MAX 3-CSP. The results of [Zwi98] were again based on *numerical evidence*.

There is an important qualitative difference between the results of [KZ97] and [Zwi98] and the other results. The $\frac{7}{8}$ ratio for MAX 3-SAT, and the $\frac{1}{2}$ ratio for MAX 3-CSP, if they are indeed obtained, are *optimal*, as follows from results of Håstad [Hås97]. It may not be too important to know whether the approximation ratio of the MAX 2-SAT algorithm of Feige and Goemans is indeed 0.931, or perhaps, due to some numerical errors it is only 0.930. It is *very* important, in our opinion, to know whether the ratio achieved for MAX 3-SAT is indeed $\frac{7}{8}$, and not, say, 0.8749. If the ratio achieved by the algorithm of [KZ97] is less than $\frac{7}{8}$, then either an improved approximation algorithm could be obtained, or a better inapproximability result could be obtained.

Unfortunately, all attempts made so far to produce an analytical proof of the $\frac{7}{8}$ conjecture failed. These attempts seem to indicate that a complete proof of the conjecture would have to consider a very large number of different cases, each corresponding to a different region of the constrained minimization problem. Even if such a complete proof is obtained, is seems that the technical details of this proof would not provide any significant insight on the nature of the MAX 3-SAT problem. Nevertheless, a proof should be obtained, if we want to put the question of the approximability threshold of the MAX 3-SAT problem to rest. It is very natural, therefore, to try to prove the conjecture using *automated means*.

As mentioned, proving that the ratio achieved by the MAX 3-SAT algorithm of [KZ97] is indeed $\frac{7}{8}$, amounts to showing that the *global* minima of a complicated non-linear constrained minimization problem in six real variables is $\frac{7}{8}$. Various numerical techniques could be used to *try* to find this global minima. (See, e.g., the optimization toolbox of Matlab [Mat96].) Most of these techniques, however, are at best guaranteed to find *local* minima. Also, most of them are subject to numerical errors. The term "numerical evidence" used above usually refers to the use of such non-rigorous techniques.

An area of research called *rigorous global optimization* (see, e.g., Hansen [Han92], Horst and Pardalos [HP95]) is devoted to obtaining *rigorous* bounds on global minima. One of the main tools used to achieve this objective is *interval arithmetic*. There are even some general purpose software packages that try to rigorously solve constrained non-linear minimization problems. Two of these are GlobSol [Kea96],[CK99] and Numerica [VMD97],[Van98]. (We note that systems like Mathematica, Matlab, Maple, do *not* provide global optimization tools.)

Unfortunately, we cannot use general purpose systems like GlobSol and Numerica, at least not directly, to obtain a proof of the $\frac{7}{8}$ and $\frac{1}{2}$ conjectures, due to several reasons. To begin with, spherical volumes are not contained in the repertoire of these systems. Even if they were, such systems, in general, can only produce *bounds* on the global minimum. Thus, it may have been possible to use such a system to show that the approximation ratio of the MAX 3-SAT algorithm of [KZ97] is, say, at least 0.8749, but not to show that it is $\frac{7}{8}$. Finally, the correctness of a proof that the ratio is at least, say, 0.8749 would depend on the correctness of the system used. GlobSol, for example, is a huge system that employs very sophisticated (rigorous) numerical techniques. Although the source code of GlobSol is freely available, checking it for correctness is an extremely difficult task. Numerica, on the other hand, is a commercial product and its source code is not even available for inspection. Furthermore, there are some further technical reasons why systems like GlobSol and Numerica could not be used directly in the analysis of MAX 3-SAT and MAX 3-CSP algorithm. Due to lack of space, we cannot elaborate on this here.

Zwick [Zwi00] uses GlobSol to analyze, and slightly optimize, the MAX 2-SAT and MAX DI-CUT approximation algorithms of Feige and Goemans. The claims of Feige and Goemans are thus verified, at least if we are willing to trust GlobSol. Note that as the approximation ratios for MAX 2-SAT and MAX DI-CUT are not expected to be 'nice' (e.g., rational) numbers, we are only interested, in any case, in bounds on them. The optimization problems to be solved do not involve spherical volumes. And, as the determination of the exact ratios obtained by the MAX 2-SAT and MAX DI-CUT algorithms is not as important as determining the ratios obtained by the MAX 3-SAT and MAX 3-CSP algorithms of [KZ97] and [Zwi98], we may be willing to accept, at least for the time being, the reliance of the correctness of the analysis on the correctness of GlobSol.

So, how can we produce checkable, rigorous proofs, of the $\frac{7}{8}$ and $\frac{1}{2}$ conjectures? We suggest here the following approach. We would first employ analytical arguments to reduce the $\frac{7}{8}$ and $\frac{1}{2}$ conjectures into claims that certain collections of constraints in real variables have *no* feasible solution. Preferably, the constraints of these collections should not involve spherical volumes. We would then use automated means to show that these collections of real constraints are indeed contradictory and have no solution.

To show that the obtained collection of real constraints are contradictory, i.e., have no real solution, we develop a system called $\mathcal{R}$EAL$\mathcal{S}$EARCH. Although $\mathcal{R}$EAL$\mathcal{S}$EARCH was written especially for this purpose, it is a fairly general system that could hopefully be used elsewhere. The main design criterion of $\mathcal{R}$EAL$\mathcal{S}$EARCH was *simplicity*. $\mathcal{R}$EAL$\mathcal{S}$EARCH uses, therefore, very naive techniques that are easily verified and is very concise, only several hundred lines of C++ code.

It is, therefore, feasible to read $\mathcal{R}$EAL$\mathcal{S}$EARCH, and verify its correctness, as part of checking the proofs of the $\frac{7}{8}$ and $\frac{1}{2}$ conjectures.

Using a combination of analytical means and $\mathcal{R}$EAL$\mathcal{S}$EARCH, we produce here rigorous proofs of the results contained in [KZ97] and [Zwi98]. There are, however, by now, some further results whose verification is even harder than the verification of the $\frac{7}{8}$ and $\frac{1}{2}$ conjectures. The analysis of the MAX 4-SAT approximation algorithm of Halperin and Zwick [HZ99], for example, involves a complicated constrained non-linear minimization problem in 10 real variables. Even more difficult would be the rigorous analysis of the MAX NAE SAT algorithm of Zwick [Zwi99], and the MAX SAT algorithm of Asano and Williamson [AW00] that is based on it. Though it should be possible, in principle, to analyze these algorithms using the techniques and tools developed here, it would involve a lot of work. As there is no reason to believe that these results are best possible (indeed, it is probably possible to improve these results, at least slightly, using the RPR^2 rounding technique introduced recently by Feige and Langberg [FL01]), it is not clear that this would be worth the effort at this stage.

3 Spherical volume inequalities

For the definition of spherical volumes, and for an explanation as to why they appear in the analysis of semidefinite programming based approximation algorithms, see [KZ97],[Zwi98]. In all the following inequalities we assume that $(\lambda_{01}, \lambda_{02}, \lambda_{12}, \lambda_{03}, \lambda_{13}, \lambda_{23})$ is a valid sequence of dihedral angles, so that $\mathrm{Vol}(\lambda_{01}, \lambda_{02}, \lambda_{12}, \lambda_{03}, \lambda_{13}, \lambda_{23})$ is well defined. In all the following inequalities, equality holds at $(\frac{\pi}{2}, \frac{\pi}{2}, \frac{\pi}{2}, \frac{\pi}{2}, \frac{\pi}{2}, \frac{\pi}{2})$, and for Inequality 3.2 also at $(\pi, \pi, \pi, \pi, \pi, \pi)$, but at no other point. (Note that $\mathrm{Vol}(\frac{\pi}{2}, \frac{\pi}{2}, \frac{\pi}{2}, \frac{\pi}{2}, \frac{\pi}{2}, \frac{\pi}{2}) = \frac{\pi^2}{8}$, and that $\mathrm{Vol}(\pi, \pi, \pi, \pi, \pi, \pi) = \pi^2$.)

INEQUALITY 3.1.

$$\mathrm{Vol}(\lambda_{01}, \lambda_{02}, \lambda_{12}, \lambda_{03}, \lambda_{13}, \lambda_{23}) \;\leq\; \frac{\pi^2}{8}$$

whenever

$$\begin{aligned} \cos\lambda_{01} + \cos\lambda_{23} + \cos\lambda_{03} + \cos\lambda_{12} &\geq 0 \\ \cos\lambda_{01} + \cos\lambda_{23} + \cos\lambda_{02} + \cos\lambda_{13} &\geq 0 \\ \cos\lambda_{02} + \cos\lambda_{13} + \cos\lambda_{03} + \cos\lambda_{12} &\geq 0 \end{aligned}$$

INEQUALITY 3.2.

$$\mathrm{Vol}(\lambda_{01}, \lambda_{02}, \lambda_{12}, \lambda_{03}, \lambda_{13}, \lambda_{23}) + \frac{7\pi^2}{32} \cdot \left(\cos\lambda_{01} + \cos\lambda_{23} + \cos\lambda_{03} + \cos\lambda_{12}\right) \;\leq\; \frac{\pi^2}{8}$$

whenever

$$\begin{aligned} \cos\lambda_{01} + \cos\lambda_{23} + \cos\lambda_{03} + \cos\lambda_{12} &\leq 0 \\ \cos\lambda_{03} + \cos\lambda_{12} - \cos\lambda_{02} - \cos\lambda_{13} &\leq 0 \\ \cos\lambda_{01} + \cos\lambda_{23} - \cos\lambda_{02} - \cos\lambda_{13} &\leq 0 \end{aligned}$$

INEQUALITY 3.3.

$$\mathrm{Vol}(\lambda_{01}, \lambda_{02}, \lambda_{12}, \lambda_{03}, \lambda_{13}, \lambda_{23}) + \frac{\pi}{2} \cdot \left(\cos\lambda_{01} + \cos\lambda_{02} + \cos\lambda_{03}\right) \;\geq\; \frac{\pi^2}{8}$$

whenever

$$\begin{aligned} \cos\lambda_{01} - \cos\lambda_{23} + \cos\lambda_{02} - \cos\lambda_{13} &\geq 0 \\ \cos\lambda_{01} - \cos\lambda_{23} + \cos\lambda_{03} - \cos\lambda_{12} &\geq 0 \\ \cos\lambda_{02} - \cos\lambda_{13} + \cos\lambda_{03} - \cos\lambda_{12} &\geq 0 \end{aligned}$$

INEQUALITY 3.4.

$$\mathrm{Vol}(\lambda_{01}, \lambda_{02}, \lambda_{12}, \lambda_{03}, \lambda_{13}, \lambda_{23}) - \frac{\pi}{6}(\lambda_{01} + \lambda_{02} + \lambda_{03}) - \frac{\pi}{3}(\lambda_{12} + \lambda_{13} + \lambda_{23}) \;\geq\; -\frac{5\pi^2}{8}$$

whenever

$$\begin{aligned} \cos\lambda_{01} - \cos\lambda_{23} &= \cos\lambda_{02} - \cos\lambda_{13} \\ &= \cos\lambda_{03} - \cos\lambda_{12} \;\leq\; 0 \end{aligned}$$

Inequality 3.1, on its own, implies that the performance ratio of the MAX 3-SAT algorithm of [KZ97] on *satisfiable* instances of the problem is $\frac{7}{8}$. Inequalities 3.1 and 3.2 together imply that the performance ratio of this algorithm for *all* instances is $\frac{7}{8}$. (As mentioned in [Zwi99], the MAX 3-SAT algorithm of [KZ97] is in fact a MAX NAE-4-SAT algorithm, and inequalities 3.1 and 3.2 imply that it achieves a performance ratio of $\frac{7}{8}$ on all instances of that problem.) Inequality 3.3 implies that the performance ratio of the MAX 3-CSP algorithm of [Zwi98] is $\frac{1}{2}$. Finally, Inequality 3.4 corresponds to the most difficult case in the proof that a second MAX 3-CSP approximation algorithm given in [Zwi98] achieves a performance ratio of $\frac{5}{8}$ on *satisfiable* instances of the problem. The other inequalities required would appear in the full version of the paper.

All attempts made so far to prove these inequalities using purely analytic tools failed. The difficulties encountered would be explained in the full version of the paper. We resort, therefore, as discussed in the previous section, to a combination of analytic and computer assisted means.

4 Proof of Inequality 3.2

We outline here the proof of Inequality 3.2. The proofs of the other volume inequalities are similar in nature.[2] The proof uses a combination of analytic and automated means.

Inequality 3.2 is equivalent to a constrained minimization problem having four constraints. The function to be minimized is $F(\lambda_{01},\lambda_{02},\lambda_{12},\lambda_{03},\lambda_{13},\lambda_{23}) = \frac{\pi^2}{8} - \mathrm{Vol}(\lambda_{01},\lambda_{02},\lambda_{12},\lambda_{03},\lambda_{13},\lambda_{23}) - \frac{7\pi^2}{32}\cdot\big(\cos\lambda_{01}+\cos\lambda_{23}+\cos\lambda_{03}+\cos\lambda_{12}\big)$. Three of the constraints are linear inequality constraints in cosines of the dihedral angles. The fourth is the requirement that $(\lambda_{01},\lambda_{02},\lambda_{12},\lambda_{03},\lambda_{13},\lambda_{23})$ be a valid sequence of dihedral angles. Boundary cases are obtained, therefore, when one, or more, of the inequality constraints are tight, or when $(\lambda_{01},\lambda_{02},\lambda_{12},\lambda_{03},\lambda_{13},\lambda_{23})$ is on the boundary of the region of valid dihedral angles. In the latter case, we say that $(\lambda_{01},\lambda_{02},\lambda_{12},\lambda_{03},\lambda_{13},\lambda_{23})$ is a degenerate sequence of dihedral angles. (This corresponds to the case in which the vectors v_0, v_1, v_2, v_3 of the semidefinite programming relaxation are linearly dependent.) This includes, in particular, cases in which $\lambda_{ij} = 0$, or $\lambda_{ij} = \pi$, for some $0 \le i < j \le 3$.

Our proof relies, among other things, on the following simple bounds on the spherical volume function. A proof of the lemma would appear in the full version of the paper.

LEMMA 4.1. *For every valid sequence of dihedral angles we have:*

$$\begin{aligned} & \frac{\pi}{2}(\lambda_{01}+\lambda_{02}+\lambda_{13}+\lambda_{23}-2\pi) \\ \le\ & \mathrm{Vol}(\lambda_{01},\lambda_{02},\lambda_{12},\lambda_{03},\lambda_{13},\lambda_{23}) \\ \le\ & \frac{\pi}{2}(\lambda_{01}+\lambda_{02}+\lambda_{03}-\pi)\ . \end{aligned}$$

Relying on Lemma 4.1 it is not difficult to prove, analytically, the following two lemmas. The detailed proofs would again appear in the full version of the paper.

LEMMA 4.2. *Inequality 3.2 holds when* $\lambda_{01},\lambda_{12},\lambda_{03},\lambda_{23} \ge \pi - \frac{16}{7\pi}$.

LEMMA 4.3. *Inequality 3.2 holds when* $(\lambda_{01},\lambda_{02},\lambda_{12},\lambda_{03},\lambda_{13},\lambda_{23})$ *is a degenerate sequence.*

[2] The proof of Inequality 3.1 is somewhat simpler. It is sketched in the extended version of [KZ97] available from `http://www.cs.tau.ac.il/~zwick/papers/max3sat-extended.ps.gz`. Almost all cases there could be handled analytically, but computer assistance is still needed at one point. In fact, it is possible to object to the proof given in [KZ97] as it relies, at that point, on a computation carried out in Mathematica. Mathematica computations (see [Wol99]), even if carried out using a large working precision, are *not* guaranteed to be correct. An alternative proof of Inequality 3.1 that relies on $\mathcal{R}$EAL$\mathcal{S}$EARCH, which *is* guaranteed to be correct, would appear in the full version of this paper.

We also have:

LEMMA 4.4. *Inequality 3.2 holds when the first inequality constraint is tight.*

This follows by noticing that when the first inequality constraint is tight, Inequality 3.2 becomes a special case of Inequality 3.1, which is already assumed to hold. (A complete proof of Inequality 3.1 would also appear in the full version of the paper.)

To finish the proof, it is enough to show that Inequality 3.2 holds at all *local minima* of the function $F(\lambda_{01},\lambda_{02},\lambda_{12},\lambda_{03},\lambda_{13},\lambda_{23})$ (defined above) that are *not* degenerate, and at which the first inequality constraint is *not* tight. (We may also assume that the conditions of Lemmas 4.2 and 4.3 are not satisfied.) Using the Fritz-John necessary conditions (see, e.g., Bertsekas [Ber99]), we get that if $(\lambda_{01},\lambda_{02},\lambda_{12},\lambda_{03},\lambda_{13},\lambda_{23})$ is such a point, then there exist $u_0,u_1,u_2 \ge 0$, not all zero, such that:

$$u_0\begin{bmatrix} \theta_{01}-\frac{7\pi^2}{16}\sin\lambda_{01} \\ \theta_{02} \\ \theta_{12}-\frac{7\pi^2}{16}\sin\lambda_{12} \\ \theta_{03}-\frac{7\pi^2}{16}\sin\lambda_{03} \\ \theta_{13} \\ \theta_{23}-\frac{7\pi^2}{16}\sin\lambda_{23} \end{bmatrix} + u_1\begin{bmatrix} 0 \\ -\sin\lambda_{02} \\ \sin\lambda_{12} \\ \sin\lambda_{03} \\ -\sin\lambda_{13} \\ 0 \end{bmatrix} + u_2\begin{bmatrix} \sin\lambda_{01} \\ -\sin\lambda_{02} \\ 0 \\ 0 \\ -\sin\lambda_{13} \\ \sin\lambda_{23} \end{bmatrix} = \begin{bmatrix} 0\\0\\0\\0\\0\\0 \end{bmatrix},$$

and such that

$$\begin{aligned} u_1(\cos\lambda_{03}+\cos\lambda_{12}-\cos\lambda_{02}-\cos\lambda_{13}) &= 0\ , \\ u_2(\cos\lambda_{01}+\cos\lambda_{23}-\cos\lambda_{02}-\cos\lambda_{13}) &= 0\ . \end{aligned}$$

In other words, $u_1 > 0$ if and only if the second inequality constraint is tight, and $u_2 > 0$ if and only if the third inequality constraint is tight. The vector multiplied by u_0 in the equation above is (twice) the gradient of the function $F(\lambda_{01},\lambda_{02},\lambda_{12},\lambda_{03},\lambda_{13},\lambda_{23})$. The partial derivatives of $\mathrm{Vol}(\lambda_{01},\lambda_{02},\lambda_{12},\lambda_{03},\lambda_{13},\lambda_{23})$ are computed using Schläfli's formula (see [Sch58],[KZ97]). The vectors multiplied by u_1 and u_2 are the gradients of the second and third inequality constraints. Note that each θ_{ij} is a rather complicated function of all the λ_{ij}'s (see, e.g., [KZ97]). Although these expressions are complicated, they do *not* involve spherical volumes, so some progress was made.

As $(\lambda_{01},\lambda_{02},\lambda_{12},\lambda_{03},\lambda_{13},\lambda_{23})$ is not degenerate, we get that $\sin\lambda_{ij} \ne 0$, for $0 \le i < j \le 3$. As not all the u_i's

can be 0, it is easy to check that we must have $u_0 > 0$. By normalizing, we may assume that $u_0 = 1$. It is also easy to check that $u_1 > 0$ or $u_2 > 0$, or both.

There are three separate cases now: (i) $u_1 > 0$ and $u_2 = 0$; (ii) $u_1 = 0$ and $u_2 > 0$; and (iii) $u_1 > 0$ and $u_2 > 0$. In case (i), the second inequality constraint is tight, in (ii), the third in tight, and in (iii) both are tight. Cases (i) and (ii) are analogous. We deal here with case (i). Case (iii) is similar and would be dealt with in the full version.

To finish the proof in case (i) it is enough to show that Inequality 3.2 holds at all the solutions of Fritz-John equations given above. Using the simple bounds on the volume given in Lemma 4.1, it is not difficult to check that this would follow from the following lemma:

LEMMA 4.5. *The following set of constraints has no real solution:*

$$\frac{\theta_{01}}{\sin\lambda_{01}} = \frac{\theta_{23}}{\sin\lambda_{23}} = \frac{7\pi^2}{16} \quad , \quad \frac{\theta_{02}}{\sin\lambda_{02}} = \frac{\theta_{13}}{\sin\lambda_{13}} \ ,$$

$$\frac{\theta_{12}}{\sin\lambda_{12}} + \frac{\theta_{13}}{\sin\lambda_{13}} = \frac{\theta_{03}}{\sin\lambda_{03}} + \frac{\theta_{13}}{\sin\lambda_{13}} = \frac{7\pi^2}{16} \ ,$$

$$\begin{aligned} &\tfrac{\pi}{2}(\lambda_{01}+\lambda_{02}+\lambda_{03}-\pi) \\ &+\tfrac{7\pi^2}{32}\cdot\big(\cos\lambda_{01}+\cos\lambda_{12}+\cos\lambda_{23}+\cos\lambda_{03}\big) \ \geq \ \tfrac{\pi^2}{8} \ , \\ &\tfrac{\pi}{2}(\lambda_{01}+\lambda_{12}+\lambda_{13}-\pi) \\ &+\tfrac{7\pi^2}{32}\cdot\big(\cos\lambda_{01}+\cos\lambda_{12}+\cos\lambda_{23}+\cos\lambda_{03}\big) \ \geq \ \tfrac{\pi^2}{8} \ , \\ &\tfrac{\pi}{2}(\lambda_{02}+\lambda_{12}+\lambda_{23}-\pi) \\ &+\tfrac{7\pi^2}{32}\cdot\big(\cos\lambda_{01}+\cos\lambda_{12}+\cos\lambda_{23}+\cos\lambda_{03}\big) \ \geq \ \tfrac{\pi^2}{8} \ , \end{aligned}$$

$$\begin{aligned} &\lambda_{01} \leq \pi - \tfrac{16}{7\pi} \text{ or } \lambda_{12} \leq \pi - \tfrac{16}{7\pi} \quad \text{or} \\ &\lambda_{03} \leq \pi - \tfrac{16}{7\pi} \text{ or } \lambda_{23} \leq \pi - \tfrac{16}{7\pi} \quad , \end{aligned}$$

$$\begin{bmatrix} 1 & -\cos\lambda_{01} & -\cos\lambda_{02} & -\cos\lambda_{12} \\ -\cos\lambda_{01} & 1 & -\cos\lambda_{03} & -\cos\lambda_{13} \\ -\cos\lambda_{02} & -\cos\lambda_{03} & 1 & -\cos\lambda_{23} \\ -\cos\lambda_{12} & -\cos\lambda_{13} & -\cos\lambda_{23} & 1 \end{bmatrix} \succeq 0 \ ,$$

$$\begin{aligned} \cos\lambda_{01}+\cos\lambda_{23}+\cos\lambda_{03}+\cos\lambda_{12} \ &\leq \ 0 \ , \\ \cos\lambda_{03}+\cos\lambda_{12}-\cos\lambda_{02}-\cos\lambda_{13} \ &= \ 0 \ , \\ \cos\lambda_{01}+\cos\lambda_{23}-\cos\lambda_{02}-\cos\lambda_{13} \ &\leq \ 0 \ , \end{aligned}$$

$$0 \leq \lambda_{01}, \lambda_{02}, \lambda_{12}, \lambda_{03}, \lambda_{13}, \lambda_{23} \leq \pi \ .$$

Some explanations are in order. First, note that in the above conditions, the θ_{ij}'s are functions of the λ_{ij}'s (see [KZ97]), and not independent variables. Second, note that some of the expressions appearing in the statement of the lemma are not defined for certain values of the λ_{ij}'s. For example, θ_{ij} may not be defined if $(\lambda_{01}, \lambda_{02}, \lambda_{12}, \lambda_{03}, \lambda_{13}, \lambda_{23})$ is not a valid sequence of dihedral angles. Similarly, division by 0 may occur when some of the λ_{ij}'s are 0. The statement of the lemma should be interpreted as follows: there is no $(\lambda_{01}, \lambda_{02}, \lambda_{12}, \lambda_{03}, \lambda_{13}, \lambda_{23})$ for which all the expressions appearing in the lemma are well defined, and for which all the specified constraints are satisfied.

An important point to note here is that is that the spherical volume function does not appear explicitly in the constraints of Lemma 4.5. Still the collection of constraints is quite complicated, and it does not seem to be an easy task to show that it has no solution. We are, however, able to obtain a computer assisted proof of Lemma 4.5 using $\mathcal{R}$EAL$\mathcal{S}$EARCH. This completes the handling of case (i). Case (ii), as we said, is almost identical. Case (iii) is dealt with in a similar manner. The details would again appear in the full version of the paper. The completes the (sketch) of the proof of Inequality 3.2.

5 Interval arithmetic

How can an automated system like $\mathcal{R}$EAL$\mathcal{S}$EARCH produce a *rigorous* mathematical proof of Lemma 4.5? The most important technique that makes this possible is *interval arithmetic*, a technique that we briefly explain here. For a more thorough treatment of interval arithmetic see, e.g., Hansen [Han92].

Almost all computers use floating point numbers to approximate real numbers. Not all real numbers are exactly representable as floating point numbers. Furthermore, even if x and y are exactly representable, their sum $x+y$, for example, not to mention expressions like $\sqrt{x}$ or $\sin y$, may not be exactly representable as a floating point number. Thus even the most basic operations on floating point numbers inevitably introduce numerical errors.

The simple solution prescribed by interval arithmetic to this predicament is to approximate each real number not by *one* floating point number, but rather by *two* of them. A real number x is approximated by an interval $[x_0, x_1]$, where $x_0 \leq x \leq x_1$, and where x_0 and x_1 are exactly representable as floating point numbers. If x is exactly representable itself, we may take $x_0 = x_1 = x$. All operations are now performed on intervals, not on single numbers. The addition of two intervals is defined as follows:

$$[x_0, x_1] + [y_0, y_1] \ = \ [\underline{x_0 + y_0}, \overline{x_1 + y_1}] \ ,$$

where $\underline{x_0 + y_0}$ is an exactly representable number that satisfies $\underline{x_0 + y_0} \leq x_0 + y_0$, and $\overline{x_1 + y_1}$ is an exactly representable number that satisfies $\overline{x_1 + y_1} \geq x_1 + y_1$. If possible, we would let $\underline{x_0 + y_0}$ be the largest floating point number satisfying $\underline{x_0 + y_0} \leq x_0 + y_0$, but this is not a firm require-

ment. Similarly, we let

$$[x_0, x_1] \cdot [y_0, y_1] = \\ \left[\min\{\underline{x_0 \cdot y_0}, \underline{x_0 \cdot y_1}, \underline{x_1 \cdot y_0}, \underline{x_1 \cdot y_1}\} , \right. \\ \left. \max\{\overline{x_0 \cdot y_0}, \overline{x_0 \cdot y_1}, \overline{x_1 \cdot y_0}, \overline{x_1 \cdot y_1}\} \right] .$$

Subtraction and division of intervals can be defined in a similar manner. (Division by an interval that contains 0 is either not allowed, causing the computation to abort, or it returns the interval $[-\infty, +\infty]$.)

In a similar way we can also define interval extensions of the elementary functions such as sqrt, sin, cos, arccos, etc. If f is a real function, then $f([x_0, x_1]) = \{f(x) \mid x_0 \le x \le x_1\}$. An interval function F is said to be an interval extension of f if and only if for every interval $[x_0, x_1]$ in the domain of f we have $f([x_0, x_1]) \subseteq F([x_0, x_1])$. More details on how interval arithmetic can actually be implemented can be found, e.g., in [Han92]. It is explained there, in particular, how numbers like $\underline{x_0 + y_0}$ and $\overline{x_1 + y_1}$ can be correctly produced using standard floating point arithmetic, and how rigorous interval extensions of functions like arccos can be obtained.

The fundamental property of interval arithmetic is that a result obtained by evaluating a real expression using interval arithmetic is *guaranteed* to be correct, in the sense that the interval obtained as a result is guaranteed to contain the correct result, even if it is not an exactly representable real number. The result obtained by such as evaluation is not always sharp, however, as the interval returned may be quite wide. This is the major drawback of using interval arithmetic. For more on the subject, see Hansen [Han92].

The use interval arithmetic is especially convenient when using an object oriented programming language like C++ (see Stroustrup [Str97]) that supports *operator overloading*. Consider, for example, a C++ expression like `(x+y)/sin(x)+x*cos(y)`. If `x` and `y` are of type `double`, then the expression is computed using double precision floating point arithmetic. If, however, `x` and `y` are variables of the class `interval` that is used to represent intervals, and if the operators `+`, `*` and `/`, and the functions `sin` and `cos` are overloaded to perform interval extensions of these operations, then the same expression is evaluated using interval arithmetic. If, for example, `x` holds the interval $[1, 2]$, and `y` holds the interval $[0, 1]$, we obtain, using a small number of inexact floating point operations, *rigorous* lower and upper bounds on the global minimum and global maximum of the function $(x + y)/\sin x + x \cos y$ in the rectangle $[1, 2] \times [0, 1]$.

6 $\mathcal{R}$EAL$\mathcal{S}$EARCH

$\mathcal{R}$EAL$\mathcal{S}$EARCH is designed to *try* to rigorously verify claims like the one appearing in Lemma 4.5. Though it was written especially for the current paper, we hope that it would find further uses. It is based, of course, on interval arithmetic and it contains, therefore, an implementation of an interval class. Many implementations of interval classes are already available. Nevertheless, we felt it necessary to implement our own class. The main objectives in the design of our class were *transparency*, we wanted to make it as easy as possible to verify the correctness of the implementation, and *portability*, we wanted the class, and $\mathcal{R}$EAL$\mathcal{S}$EARCH, to compile and work correctly on almost any combination of C++ compiler and machine architecture. We also wanted to make as few assumptions as possible on the underlying floating point arithmetic of the processor used.

$\mathcal{R}$EAL$\mathcal{S}$EARCH is a very general, yet very naive, tool. It is designed to *try* to rigorously verify claims like:

> No $(x_1, x_2, \ldots, x_n)$, where $a_i \le x_i \le b_i$, for $1 \le i \le n$, satisfies all the following conditions:
>
> $$f_1(x_1, x_2, \ldots, x_n) \ge 0 ,$$
> $$f_2(x_1, x_2, \ldots, x_n) \ge 0 \text{ or } f_3(x_1, x_2, \ldots, x_n) \ge 0$$
> $$\ldots$$

The functions $f_1(x_1, x_2, \ldots, x_n)$, $f_2(x_1, x_2, \ldots, x_n)$ and $f_3(x_1, x_2, \ldots, x_n)$ here are, say, elementary functions defined using the basic arithmetic operations, the trigonometric and inverse trigonometric functions, etc. $\mathcal{R}$EAL$\mathcal{S}$EARCH *tries* to verify such claims but it is *not* guaranteed to succeed, even if the claim is true. (Indeed, the task of verifying such claims may not be recursively enumerable.) There are, however, some cases in which $\mathcal{R}$EAL$\mathcal{S}$EARCH is guaranteed to succeed. And, proofs produced by $\mathcal{R}$EAL$\mathcal{S}$EARCH are always correct.

$\mathcal{R}$EAL$\mathcal{S}$EARCH works in the following way. It lets $X_1 = [a_1, b_1]$, $X_2 = [a_2, b_2]$, ..., $X_n = [a_n, b_n]$. (For simplicity, we assumed here that the a_i's and b_i's are exactly representable. If not, they are replaced by $\underline{a_i}$ and $\overline{b_i}$, for $1 \le i \le n$.) It then evaluates $Y_j = F_j(X_1, X_2, \ldots, X_n)$, for $1 \le j \le k$, where F_j is the interval extension of f_j obtained by evaluating the expression defining f_j using interval arithmetic. If $Y_1 < 0$, i.e., if Y_1 contains only negative numbers, or if $Y_2 < 0$ and $Y_3 < 0$, etc., the claim is verified!

Of course, this very naive attempt to verify the claim would usually fail. If it does, $\mathcal{R}$EAL$\mathcal{S}$EARCH uses the divide and conquer technique. It chooses an index i, $1 \le i \le n$, e.g., the one for which $b_i - a_i$ is maximized, and breaks X_i into two smaller intervals $X_i' = [a_i, \overline{(a_i + b_i)/2}]$ and $X_i'' = [\underline{(a_i + b_i)/2}, b_i]$. Note that the two intervals may slightly overlap, but they always cover the original interval X_i. It then tries to verify recursively the two corresponding subclaims. This process may eventually terminate, thereby succeeding in verifying the original claim, or may run for ever. ($\mathcal{R}$EAL$\mathcal{S}$EARCH is instructed, of course, to give up

after a certain amount of time, or at a certain depth of the recursion.)

We did not specify exactly, up till now, the exact form of the claims that $\mathcal{R}$EAL$\mathcal{S}$EARCH can try to verify. This is because $\mathcal{R}$EAL$\mathcal{S}$EARCH may try to verify any claim that can be expressed in C++! More specifically, the 'input' to $\mathcal{R}$EAL$\mathcal{S}$EARCH is composed of: (i) n – the number of variables; (ii) The bounds a_i and b_i, for $1 \leq i \leq n$; (iii) a C++ procedure, written by the user of $\mathcal{R}$EAL$\mathcal{S}$EARCH, that accepts a vector of n intervals and returns 'true' if and only if the procedure succeeded in automatically verifying the sub-claim corresponding to the intervals passed to it as arguments. Usually, the user's procedure just tries to verify the sub-claim naively using interval arithmetic, but the user is free to use more sophisticated means. The correctness of the proof produced depends, of course, on the correctness of the procedure supplied by the user.

$\mathcal{R}$EAL$\mathcal{S}$EARCH, therefore, is quite a simple system. It supplies the user with a rigorous implementation of interval arithmetic, and it handles the divide and conquer process for her. All the rest is up to the user. This simplicity makes $\mathcal{R}$EAL$\mathcal{S}$EARCH a very flexible tool. A complete description of $\mathcal{R}$EAL$\mathcal{S}$EARCH would appear in the full version of the paper.

One technical point that should be explained, to give the reader a full understanding of how $\mathcal{R}$EAL$\mathcal{S}$EARCH should be *rigorously* used, is the following. The computation of an interval extension of an expression may sometime involve ill-behaved operations like dividing by an interval that contains 0, or taking a root of an interval of negative numbers. It is the user's responsibility to deal correctly with such situations. One correct way of dealing with a situation like this is to abandon the evaluation of the expression in which the situation occurred and to assume that the constraint in which this expression appears *may* be satisfied by a point in the considered box. In some cases, less drastic actions may also be justified.

As can be seen, $\mathcal{R}$EAL$\mathcal{S}$EARCH is very naive. Yet, it is extremely powerful. $\mathcal{R}$EAL$\mathcal{S}$EARCH, with an appropriate user supplied procedure, verifies the correctness of Lemma 4.5 in less than a minute, checking about 500,000 sub-claims in the process. The relation of $\mathcal{R}$EAL$\mathcal{S}$EARCH to existing rigorous global optimization tools such as GlobSol [Kea96],[CK99] and Numerica [VMD97],[Van98] would be discussed in the full version of this paper.

7 Concluding remarks

We obtained computer assisted proofs of the $\frac{7}{8}$ and $\frac{1}{2}$ conjectures of [KZ97] and [Zwi98]. The proofs were obtained with the help of $\mathcal{R}$EAL$\mathcal{S}$EARCH, a simple, yet powerful, tool. $\mathcal{R}$EAL$\mathcal{S}$EARCH was deliberately kept as simple as possible, to make it easier to verify its correctness. The source code of $\mathcal{R}$EAL$\mathcal{S}$EARCH, and of the specific procedures used to verify Lemma 4.5 and the other required lemmas, should be considered as part of our proofs. (They are all available from `http://www.cs.tau.ac.il/~zwick/RealSearch.html`. We encourage the reader to read and execute them.)

But, are these really proofs? A typical computer assisted proof, like ours, is essentially composed of two steps. The first step says something like: "Here is program P. If program P, when executed by an abstract computer, outputs YES, then the theorem is true, because ...". The second step says something like: "I ran program P and it said YES!", or more specifically: "I compiled program P using compiler A, under operating system B, and ran it on processor C, again under operating system B. It said YES!".

The first step is a conventional mathematical proof. It simply argues that a certain program has certain properties. Such arguments are common in computer science. Anyone who objects to this step of the proof should find a flaw, or a gap, in the arguments made.

The second step is more problematic. It is certainly *not* a proof in the conventional mathematical sense of the word. Many things can go wrong here. Program P may not have been compiled correctly by compiler A. Or, due to some bug in operating system B, the program did not run as intended. Or, processor C may suffer from a design flaw that causes an incorrect execution of the program, or perhaps the specific processor used has a short-circuit somewhere, and so on. It is possible to reduce the likelihood of such problems by compiling the program using several different compilers, and running it on different processors. But, while we may eventually be able to produce mathematical correctness proofs for compilers, operating systems, and the hardware design of the processors, it seems that we would never be able to produce a mathematical proof that no hardware fault occurred during a specific computation.

What we can do, is instruct the computer executing program P to print a trace of the execution. This trace is a mathematical proof, though perhaps a not so inspiring one. Like any mathematical proof it should be checked carefully to make sure that it is correct. In many cases, however, this is not humanly possible.[3] The main problem with computer assisted proofs, therefore, is that they are usually too *long*. They are also, in most cases, less insightful than conventional proofs. (A separate discussion, that we cannot enter here, can focus on whether computer assisted proofs are less, or more, trustworthy than conventional proofs.)

[3] We note in passing that PCPs may come to the rescue here, though, in principle, the same problems remain, and the use of randomness in the verification of proofs is in itself a deviation from the traditional notion of a proof.

Nevertheless, we feel that the use of the word 'proof', as made in this paper, is justified. We use it as a shorthand for 'computer assisted proof' which is taken to include all the qualifications made above.

Next, a note on floating point arithmetic. While most people have grown accustomed to the assumption that computers perform integer arithmetic correctly, some still find it hard to accept that they perform floating point arithmetic correctly, i.e., as specified by the IEEE-754 standard. The implementation of floating point arithmetic is indeed harder than the implementation of integer arithmetic, but it is not much harder, so there is no essential difference between the assumption that the integer arithmetic of a computer is correct and that its floating point arithmetic is correct.

We predict that sophisticated versions of $\mathcal{R}$EAL$\mathcal{S}$EARCH-like tools would be in general use in the not so distant future, and that using them would be viewed in much the same way as we now view the use of calculators.

References

[AH77] K. Appel and W. Haken. Every planar map is four colorable. Part I. Discharging. *Illinois Journal of Mathematics*, 21:429–490, 1977.

[AHK77] K. Appel, W. Hakenn, and J. Koch. Every planar map is four colorable. Part II. Reducibility. *Illinois Journal of Mathematics*, 21:491–597, 1977.

[ALM+98] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45:501–555, 1998.

[ANS85] ANSI/IEEE. Standard 754-1985 for binary floating-point arithmetic. 1985. Reprinted in *ACM SIGPLAN Notices*, 22(2):9–25, 1987.

[AS98] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45:70–122, 1998.

[AW00] T. Asano and D.P. Williamson. Improved approximation algorithms for MAX SAT. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, California*, pages 96–105, 2000.

[Ber99] D.P. Bertsekas. *Nonlinear Programming*. Athena Scientific, second edition, 1999.

[BGS98] M. Bellare, O. Goldreich, and M. Sudan. Free bits, PCPs, and nonapproximability—towards tight results. *SIAM Journal on Computing*, 27:804–915, 1998.

[CK99] G. Coreliss and R.B. Kearfott. *Global Solutions User Guide (draft)*. Currently available from `http://www.mscs.mu.edu/~globsol/User_Guide/index.html`, 1999.

[FG95] U. Feige and M.X. Goemans. Approximating the value of two prover proof systems, with applications to MAX-2SAT and MAX-DICUT. In *Proceedings of the 3nd Israel Symposium on Theory and Computing Systems, Tel Aviv, Israel*, pages 182–189, 1995.

[FGL+96] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM*, 43:268–292, 1996.

[FL01] U. Feige and M. Langberg. The RPR^2 rounding technique for semidefinite programs. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming, Crete, Greece*, pages 213–224, 2001.

[GMT00] D. Gabai, R. Meyerhoff, and N. Thurston. Homotopy hyperbolic 3-manifolds are hyperbolic, 2000. To appear in *Annals of Mathematics*.

[Gol91] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23:5–48, 1991.

[GW95] M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42:1115–1145, 1995.

[Hal97a] T. C. Hales. Sphere packings. I. *Discrete & Computational Geometry*, 17(1):1–51, 1997.

[Hal97b] T. C. Hales. Sphere packings. II. *Discrete & Computational Geometry*, 18(2):135–149, 1997.

[Hal98] T.C. Hales. The Kepler conjecture, 1998. Mathematics arXiv `http://arxiv.org/abs/math.MG/9811078`.

[Hal00] T.C. Hales. Cannonballs and honeycombs. *Notices of the American Mathematical Society*, 47(4):440–449, 2000.

[Hal01] T.C. Hales. The honeycomb conjecture. *Discrete & Computational Geometry*, 25(1):1–22, 2001.

[Han92] E. Hansen. *Global optimization using Interval Analysis*. Marcel Dekker, 1992.

[Hås97] J. Håstad. Some optimal inapproximability results. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing, El Paso, Texas*, pages 1–10, 1997. Full version available as E-CCC Report number TR97-037.

[Hau98] J. Hauser. SoftFloat. `http://www.cs.berkeley.edu/~jhauser/arithmetic/softfloat.html`, 1998.

[HM98] T.C. Hales and S. McLaughlin. A proof of the dodecahedral conjecture, 1998. Mathematics arXiv `http://arxiv.org/abs/math.MG/9811079`.

[HP95] R. Horst and P.M. Pardalos, editors. *Handbook of global optimization*. Kluwer, 1995.

[HS00] J. Hass and R. Schlafly. Double bubbles minimize. *Annals of Mathematics. Second Series*, 151(2):459–515, 2000.

[HZ99] E. Halperin and U. Zwick. Approximation algorithms for MAX 4-SAT and rounding procedures for semidefinite programs. In *Proceedings of the 7th Conference on Integer Programming and Combinatorial Optimization, Graz, Austria*, pages 202–217, 1999.

[Kah96a] W. Kahan. Lecture notes on the status of IEEE standard 754 for binary floating-point arithmetic. `http://http.cs.berkeley.edu/~wkahan/ieee754status/ieee754.ps`, 1996.

[Kah96b] W. Kahan. What can you learn about floating-point arithmetic in one hour? `http://http.cs.berkeley.edu/~wkahan/ieee754status/cs267fp.ps`, 1996.

[Kea96] R.K. Kearfott. *Rigorous global search: continuous problems*. Kluwer, 1996.

[KZ97] H. Karloff and U. Zwick. A $7/8$-approximation algorithm for MAX 3SAT? In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, Miami Beach, Florida*, pages 406–415, 1997.

[Mat96] The MathWorks Inc. *MATLAB optimization toolbox, User's guide, Version 5*, 1996.

[Ove01] M.L. Overton. *Numerical Computing With IEEE Floating Point Arithmetic*. SIAM, 2001.

[Raz98] R. Raz. A parallel repetition theorem. *SIAM Journal on Computing*, 27:763–803, 1998.

[RSST97] N. Robertson, D. Sanders, P. Seymour, and R. Thomas. The four-colour theorem. *Journal of Combinatorial Theory, Series B*, 70(1):2–44, 1997.

[Sch58] L. Schläfli. On the multiple integral $\int^n dx\,dy\ldots dz$, whose limits are $p_1 = a_1x + b_1y + \ldots + h_1z > 0, p_2 > 0, \ldots, p_n > 0$, and $x^2+y^2+\ldots+z^2 < 1$. *Quarterly Journal of Mathematics (Oxford)*, 2:269–300, 1858. Continued in Vol. 3 (1860), pp. 54–68 and pp. 97-108.

[Str97] B. Stroustrup. *The C++ programming language*. Addison-Wesley, third edition, 1997.

[Van98] P. Van Hentenryck. A gentle introduction to Numerica. *Artificial Intelligence*, 103:209–235, 1998.

[VMD97] P. Van Hentenryck, L. Michel, and Y. Deville. *Numerica – A modeling language for global optimiztion*. MIT Press, 1997.

[Wol99] S. Wolfram. *The Mathematica book*. Cambridge University Press, fourth edition, 1999.

[Zwi98] U. Zwick. Approximation algorithms for constraint satisfaction problems involving at most three variables per constraint. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, California*, pages 201–210, 1998.

[Zwi99] U. Zwick. Outward rotations: a tool for rounding solutions of semidefinite programming relaxations, with applications to max cut and other problems. In *Proceedings of the 31th Annual ACM Symposium on Theory of Computing, Atlanta, Georgia*, pages 679–687, 1999.

[Zwi00] U. Zwick. Analyzing the MAX 2-SAT and MAX DICUT approximation algorithms of Feige and Goemans. Submitted for publication, 2000.

A Floating-point arithmetic and the IEEE standard

We shortly sketch here the features of the IEEE-754 standard of floating point arithmetic that are most important for our purposes. A complete description of the standard appears in [ANS85]. For further explanations of the standard, and the rational behind some of its features, see Goldberg [Gol91], Kahan [Kah96b],[Kah96a]), and the book of Overton [Ove01]. As mentioned, most commercially available processors conform to the standard, the most notable exception is the CRAY processor.

The standard specifies several different floating point formats. The most common one, and the one we use, is the 64-bit format. This format is used by most C and C++ compilers for variables of type `double`. Each 64-bit floating point number is composed of: (i) a sign bit; (ii) an 11-bit exponent; and (iii) a 52-bit significant (also known as mantissa). The significant has an implicit leading 1 bit, unless the number is *denormalized* (see the references above for an explanation). The smallest positive number that can be represented in this format is $2^{-1074} \simeq 4.941 \times 10^{-324}$, the smallest normalized number is $2^{-1022} \simeq 2.225 \times 10^{-308}$, and the largest finite number representable is $(2 - 2^{-52})2^{1023} \simeq 1.798 \times 10^{308}$. There are also special bit patterns that represent $+0$, -0, $+\infty$ (plus infinity), $-\infty$ (minus infinity), and `NaN` (not a number). (Note that, for some good reasons, $+0$ and -0 are different entities, though they behave identically in almost all operation.)

The most important feature of the standard from our point of view are the following *rounding modes* that it mandates: (i) Round to nearest (default); (ii) Round towards $+\infty$; (iii) Round towards $-\infty$; (iv) Round towards 0. According to the standard, the result of each addition, subtraction, multiplication, division, and square root operation on floating point arguments should be equal to the result obtained by computing the *exact* mathematical result of the operation, even if it is not representable as a floating point number, and rounding it to the nearest floating point number in the appropriate direction. In the first rounding mode, if the exact result falls exactly between two floating point numbers, then the one with a 0 least significant bit is chosen.

In other words, the standard specifies that the result of each addition, subtraction, multiplication, division, and square root operation should be *correctly rounded*. A numerical error is introduced only if the exact result is not representable as a floating point number. Furthermore, the user is allowed to choose the rounding mode used, and he/she may change it before each operation. The standard does *not* say anything regarding the evaluation of elementary functions like $\sin$, $\cos$, $\arcsin$, $\arccos$, $\exp$, $\log$, etc.

The standard also specifies the meaning of operations involving $\pm\infty$ and `NaN`'s. Thus, for example, $(+\infty) + (+\infty) = (+\infty)$, $(+\infty) \cdot (+\infty) = (+\infty)$, but $(+\infty) - (+\infty) =$ `NaN`. Also, $(\pm 0)/(\pm 0) =$ `NaN`, but $1/(+0) = +\infty$, $1/(-0) = -\infty$. All these definitions follow the usual mathematical conventions. Every numerical operation involving a `NaN` returns a `NaN`.

Implementing correctly rounded floating point arithmetic, as prescribed by the standard, is not a difficult task. For more details, see the references given above.

The floating point standard provides an ideal setting for the implementation of interval arithmetic. There are, however, two main obstacles to be overcome. The first is that most programming languages do not provide commands for changing rounding modes. The second is that the standard is silent regarding functions like $\sin$, $\cos$ and $\arccos$. The ways in which these obstacles are overcome would be described in the full version of the paper.