Approximation of constraint satisfaction problems

Johan Håstad

KTH Numerical Analysis
and Computer Science

May 17, 2008

Questions please!

We are from different fields so our perspectives are probably different.

Interaction is a great invention.

Please ask questions!

Disclaimer

Will tell a story, few technical details and not many references.

Will not live up to title and hardly get beyond 3-Sat.

My view of the world

Complexity theory.

My view of the world

Complexity theory.

Solvable means efficiently solvable which is solvable in (probabilistic) polynomial time (as measured of inputs size).

My view of the world

Complexity theory.

Solvable means efficiently solvable which is solvable in (probabilistic) polynomial time (as measured of inputs size).

NP-complete problems are hard, i.e. NP$\neq$P.

My view of the world

Complexity theory.

Solvable means efficiently solvable which is solvable in (probabilistic) polynomial time (as measured of inputs size).

NP-complete problems are hard, i.e. NP$\neq$P.

Interested in provable properties.

Favorite problem

Our favorite problem is 3-Sat

$$(\bar{x}_1 \vee x_7 \vee x_{11}) \wedge \ldots (x_4 \vee \bar{x}_9 \vee x_{25})$$

Most of the time not random.

Each clause of length 3.

Usually n variables, m clauses.

Basic approach

NP-complete, cannot solve all instances efficiently.

Basic approach

NP-complete, cannot solve all instances efficiently.

Let us look into algorithms that do reasonably well for each input.

Approximation ratio

We define

$$\alpha = \frac{\text{Number of satisfied constraints}}{\text{Optimal number of satisfied constraints}}$$

worst case over inputs, expected over internal randomness if we have a probabilistic algorithm.

Trivial approximation ratio

A random assignment satisfies on the average $7m/8$ clauses (also easy to do deterministically).

No assignment satisfies more than all m clauses.

Trivial approximation ratio

A random assignment satisfies on the average $7m/8$ clauses (also easy to do deterministically).

No assignment satisfies more than all m clauses.

We get an approximation ratio of $7/8$ which is the trivial approximation ratio.

My favorite problem

Similarly we get a trivial approximation ratio for other problems. Comparing the number of constraints satisfied by a random assignment to all constraints.

For which type of constraints can we do better?

A problem on randomized Sat

Suppose we have random 3-Sat formula, fairly dense and hence probably not satisfiable.

We want a polynomial time algorithm that is allowed to answer "satisfiable", "not satisfiable" and "don't know" and never is allowed to be incorrect.

For what values of m can the algorithm decide most formulas?

Results so far

An approximation algorithm with α a constant greater than $7/8$ would give a positive result for any $m = \omega(n)$.

This follows as for such m the best assignment satisfies $(7/8 + o(1))m$ clauses.

State of knowledge for this problem

Best results handle $m = c \cdot n^{3/2}$, no consensus of correct answer.

Nice problem to think about!

Minimal hope for approximation

A promise problem for some $\epsilon > 0$.

Given a formula φ, I guarantee that it is either satisfiable or no assignment satisfies more than $(7/8 + \epsilon)m$ clauses. Can you tell which?

Minimal hope for approximation

A promise problem for some $\epsilon > 0$.

Given a formula φ, I guarantee that it is either satisfiable or no assignment satisfies more than $(7/8 + \epsilon)m$ clauses. Can you tell which?

Theorem [H]: For any $\epsilon > 0$ it is NP-hard to tell which of the two is the case.

Weaker formulation

Theorem: For any $\epsilon > 0$ it is NP-hard to approximation Max-3-Sat within $7/8 + \epsilon$.

It is NP-hard to beat the trivial approximation ratio.

Max-3-Sat is approximation resistant.

NP-hard?

A problem is NP-hard if solving it in polynomial time implies that NP=P.

Thus the same property as being NP-complete, but we do not require the problem to belong to NP.

Proving the theorem

Proof on high level. Given a formula φ we, in polynomial time, produce a formula ψ such that.

If φ is satisfiable so is ψ.

If φ is not satisfiable then no assignment satisfies more than a fraction $7/8 + \epsilon$ of the clauses of ψ.

In short

The formula ψ satisfies the promise and distinguishing the two cases is equivalent to determining whether φ is satisfiable.

An algorithm that solves the promise problem also solves satisfiability by reduction.

In short

The formula ψ satisfies the promise and distinguishing the two cases is equivalent to determining whether φ is satisfiable.

An algorithm that solves the promise problem also solves satisfiability by reduction.

This is essentially the definition of being NP-hard.

In short

The formula ψ satisfies the promise and distinguishing the two cases is equivalent to determining whether φ is satisfiable.

An algorithm that solves the promise problem also solves satisfiability by reduction.

This is essentially the definition of being NP-hard.

Creating ψ is a long story, let us tell a tiny part.

Checking proofs

A proof is checked by a (polynomial time) verifier V.

In a standard proof V reads the entire proof and is certain of its validity.

To be concrete

Instance: A formula φ, claimed to be satisfiable.

Proof: A satisfying assignment to φ.

Checking: Substituting the assignment in φ and making sure that it is satisfied.

Properties

V runs in deterministic polynomial time.

V reads the entire proof which is of polynomial size.

V is never fooled to accept an incorrect claim or an incorrect proof of a correct claim.

Probabilistically Checkable Proofs

We want to formalize a notion where the verifier only does random spot-checks and reads very few of the bits.

Probabilistically Checkable Proofs

We want to formalize a notion where the verifier only does random spot-checks and reads very few of the bits.

V can be fooled but with small probability.

Probabilistically Checkable Proofs

We want to formalize a notion where the verifier only does random spot-checks and reads very few of the bits.

V can be fooled but with small probability.

This is a Probabilistically Checkable Proof (PCP).

Remember our reduction

Given φ we efficiently produce ψ such that:

If φ is satisfiable so is ψ.

If φ is not satisfiable then no assignment satisfies more than a fraction $7/8 + \epsilon$ of the clauses of ψ.

Remember our reduction

Given φ we efficiently produce ψ such that:

If φ is satisfiable so is ψ.

If φ is not satisfiable then no assignment satisfies more than a fraction $7/8 + \epsilon$ of the clauses of ψ.

A (satisfying) assignment to the variables of ψ gives a PCP!

The basic PCP

Given ψ, satisfiable or at most $(7/8+\epsilon)$ satisfiable.

Proof: A satisfying assignment.

Checking: Pick a random clause and read the three variables in to see if it satisfied.

The basic PCP

Given ψ, satisfiable or at most $(7/8 + \epsilon)$ satisfiable.

Proof: A satisfying assignment.

Checking: Pick a random clause and read the three variables in to see if it satisfied.

Completeness 1, soundness $7/8 + \epsilon$, reading 3 bits.

Getting this ultimate PCP

Obviously (I hope) our final reduction has magical properties on top of proving the theorem.

Would be too much to hope for to simply write it down and check in a couple of slides.

The PCP-theorem

The PCP theorem [Arora, Lund, Motwani, Sudan and Szegedy, 1990]: It is possible to check satisfiability by a polynomial size proof, that reads $O(1)$ bits, has completeness 1 and soundness $1/2$.

The PCP-theorem

The PCP theorem [Arora, Lund, Motwani, Sudan and Szegedy, 1990]: It is possible to check satisfiability by a polynomial size proof, that reads $O(1)$ bits, has completeness 1 and soundness $1/2$.

Gives the qualitatively correct statement but constants originally were not so good.

Proving the PCP-theorem

A complicated construction, using many tools.

- Coding an assignment by a lower degree polynomial over a larger domain.
- Properties of low-degree polynomials. Testing that a table is a low degree function.
- Recursive techniques, a proof that there is a proof that there is a proof.

New proof by Dinur

Obtained in 2006.

A (mostly) combinatorial proof relying on recursive construction with expander graphs.

Builds on experience in designing PCPs accumulated over time.

Getting good constants

Additional levels of recursion.

Using interesting codings of Boolean strings, the long code which is the value of all Boolean functions of the input, i.e. as

$$f(x) : f\{0,1\}^t \mapsto \{0,1\}.$$

Getting good constants

Additional levels of recursion.

Using interesting codings of Boolean strings, the long code which is the value of all Boolean functions of the input, i.e. as

$$f(x) : f\{0,1\}^t \mapsto \{0,1\}.$$

Coding t bits as 2^{2^t} bits, where t is some constant.

A key analysis tool

Use the Fourier transform to analyze Boolean functions.

Early results use simple properties of the Fourier transform.

Recent results rely on strong results in the real domain.

A small piece

Given a big table, T, how to we verify that it codes a linear function $\{0,1\}^t \mapsto \{0,1\}$?

A small piece

Given a big table, T, how to we verify that it codes a linear function $\{0,1\}^t \mapsto \{0,1\}$?

Pick random x and y and check if $T(x) + T(y) = T(x+y)$ [BLR].

A small piece

Given a big table, T, how to we verify that it codes a linear function $\{0,1\}^t \mapsto \{0,1\}$?

Pick random x and y and check if $T(x) + T(y) = T(x + y)$ [BLR].

If test accepts with probability $1 - \delta$ then exists linear function L that that $T(x) = L(x)$ for a fraction $1 - \delta$ fraction of x.

Checking other properties

Checking that a table is a low degree polynomial.

Checking that different tables code related strings or string with certain properties.

Checking other properties

Checking that a table is a low degree polynomial.

Checking that different tables code related strings or string with certain properties.

Entering these constructions would take us to far..

Other types of constraints

Can prove similar results for other constraints.

Given linear equations mod p for a prime p it is NP-hard to distinguish systems where we can satisfy a fraction $1-\epsilon$ of the equations from those where we can only satisfy a fraction $1/p+\epsilon$.

Final words

Exists many inapproximability results.

Many predicates are approximation resistant.

Many predicates are not, key technique is semidefinite programming.

Interesting proofs that can be checked very efficiently.