Complexity Classes IV

NP Optimization Problems and Probabilistically Checkable Proofs

Eric Rachlin

Decision vs. Optimization

- Most complexity classes are defined in terms of Yes/No questions.
- In the case of NP, we wish to know if a certificate exists that satisfies certain constraints (i.e. SAT, vertex cover, clique, …)
- Even if no certificate exists, we can still ask how many constraints can be satisfied, or how large (or small) some parameter can be.
- We let **OPT** to denote this value.

Decision vs. Optimization

- With respect to polynomial time, optimization is no harder than decision
- Example: MAXCLIQUE (perform binary search over instances of CLIQUE)
- Example: MAXSAT (perform binary search using a variant of SAT that asks if k clauses can be satisfied)

Approximation Algorithms

- If P $\neq$ NP, we cannot find OPT for an NP-complete optimization problem in polynomial time (PTIME).
- In practice, we may not need an exact answer (particularly if the parameters of the problem are themselves estimates).
- An **approximation algorithm** computes OPT' such that $|OPT - OPT'| \leq f(OPT)$ for some f.
- For NP-complete problems, can f(OPT) be arbitrarily small?

What can we hope for?

- A **Polynomial Time Approximation Scheme (PTAS)** for an optimization problem is an algorithm that, for a given ε, results in a PTIME approximation algorithm such that $|OPT - OPT'| \leq \varepsilon OPT$.
- The approximation algorithm can still have a runtime that is exponential in $1/\varepsilon$.
- **Efficient Polynomial Time Approximation Scheme (EPTAS)** adds the requirement that the runtime be of the form $f(\varepsilon)$*poly(N).

How good is too good?

- A **Fully Polynomial Time Approximation Scheme (FPTAS)** is PTAS where the running time of the approximation algorithm is also polynomial in $1/\varepsilon$.
- It is not hard to show that an FPTAS for some NP-complete problems implies P = NP.
- It turns out the same is true for a PTAS, but this is far from obvious. It is a consequence of the PCP Theorem.

Strongly NP-Hard Problems

- A problem is strongly NP-hard if its NP-hardness does not require any of its numerical parameters to be exponential in the length of the problem.
- Examples: CLIQUE, TSP, SAT, …
- If an FPTAS exists for CLIQUE, we can approximate the solution to a factor less than 1/N and obtain an exact solution.

Hardness of Approximation

- Do PTASs exist for strongly NP-hard problems?
  - Yes!
  - Examples: Planar TSP, Euclidian TSP
- How can we show a PTAS does not exist for certain NP-complete problems?
  - Define NP in terms of PCPs…
  - …this leads to a gap introducing reduction…
  - …which leads to gap preserving reductions.

Reductions and NP

- Recall Cook's Theorem (1971):
  - SAT is NP-Complete
  - The "tableau" of a nondeterminstic Turing machine can be converted to an instance of SAT.
  - The instance of SAT is polynomial in the size of the tableau, and is satisfied if and only in the tableau accepts (and is valid).
- SAT was then reduced to other NP-complete problems (Karp, 1972).

More on Cook's Theorem

- It is easy to show that the following language, ACCEPT, is NP-complete:
  - Let <M, x, 1^t> be a triple consisting of a deterministic Turing machine, a binary input to M, and a string of t 1's.
  - <M, x, 1^t> is in the language if M accepts some string of the form <x, y> in at most t steps. (Here y represents a certificate of length at most t.)
- To prove Cook's Theorem, give a polynomial time algorithm that designs a circuit outputting 1 if and only if M accepts <x, y> after t steps.

So what needs work?

- In Cook's Theorem, the instance of SAT is satisfiable iff the nondeterministic Turing machine accepts after poly(N) steps.
- Even when it does not accept, the instance of SAT is still "almost" satisfiable.
- We want to introduce a gap.
  - Either the instances of SAT are satisfiable,
  - Or some fixed fraction of clauses are unsatisfied by any assignment of values to variables.

Gap Introducing Reduction

- Let x be an instance of some NP-complete decision problem L, let L(x) denote *Is x in L?*
- Let MAXL(x) be the corresponding optimization problem.
- A polynomial time (PTIME) reduction from L to L' is some PTIME function, R, such that L'(R(x)) = L(x).
- R is **gap introducing** if, for all L(x) = 1 and L(y) = 0, MAXL'(R(x))/MAXL'(R(y)) $\geq \Delta$.

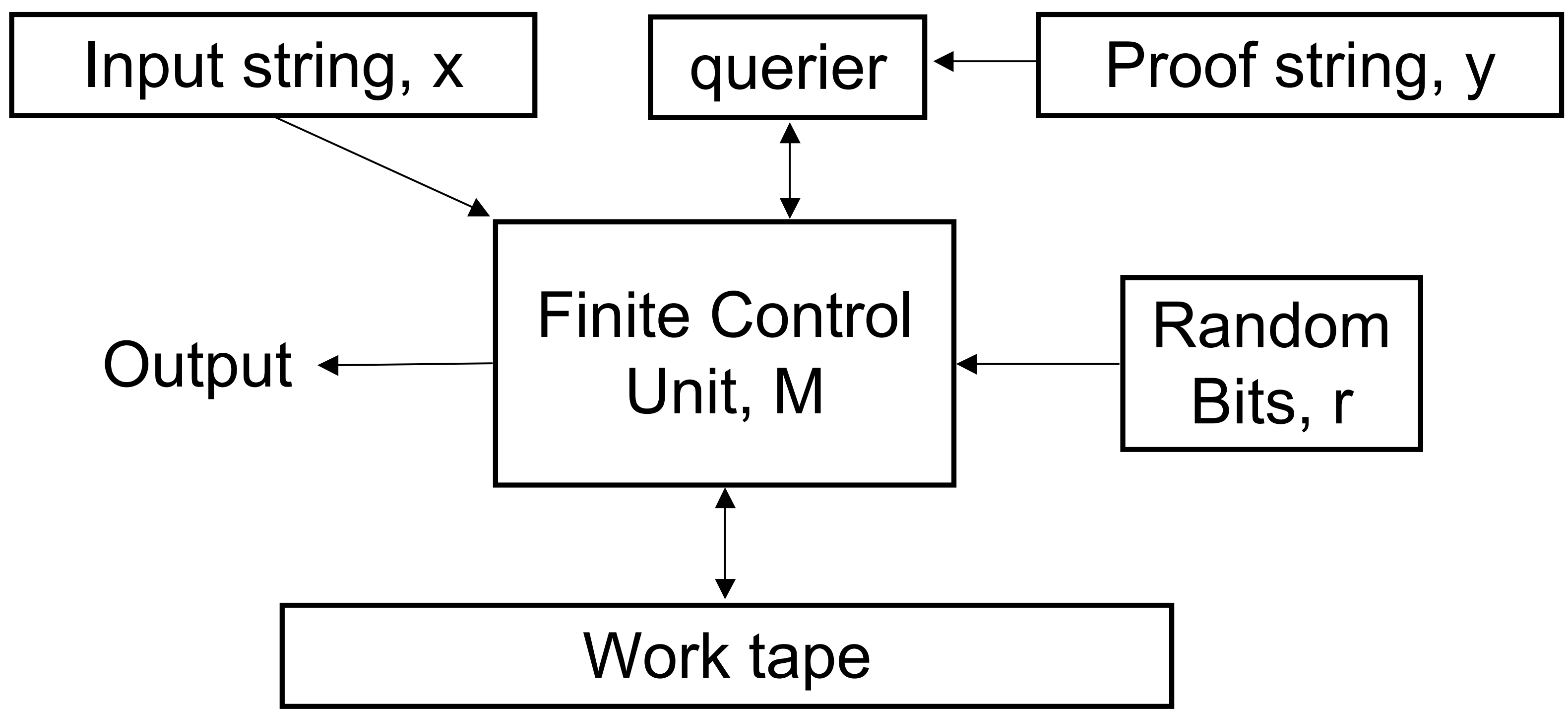
Gap Preserving Reductions

- If L is NP-complete, L' is in NP, and R(x) is a PTIME gap introducing reduction from L to L':
  - L' is NP-complete
  - MAXL' is inapproximable to within a factor of Δ (if $P \neq NP$).
- Let R' be a reduction from L' to L''. R' is **gap preserving** if there exists a constant ß such that for any constant Δ
  - if MAXL'(x)/MAXL'(y) $\geq \Delta$
  - then MAXL''(R(x))/MAXL''(R(y)) $\geq$ ß
- If MAXL' is inapproximable to within a factor of Δ, R' shows that L'' is inapproximable to within a factor ß.

Going from NP to PCP

- Nondeterminism is equivalent to having access to a polynomial-sized "certificate".
  - If a valid certificate exists, the machine accepts.
  - We see that many problems which appear hard to solve are easy to check.
- For PCPs, machines also have access to a certificate (called a proof).
  - The proof is selectively queried using random bits.
  - A valid proof causes the machine to accept, an invalid proof will be rejected with high probability.

Machines with access to random bits and a proof



Randomized Computation

- Random bits allow machines to recognize languages with high probability (w.h.p.)
  - Example: Polynomial Identity Testing.

- **Completeness** is the probability of recognizing a string in the language.
- **Soundness** is the probability of accepting a string not in the language.

Completeness and Soundness with Certificates

- For a TM accepting a language L, with access to random bits and a proof/certificate:
  - Completeness c means that there exists a certificate such that strings in L are accepted with probability c.
  - Soundness s means that for all certificates the TM accepts strings not in L with probability s.

PCP Complexity Classes

- **$PCP_{c,\, s}$[q(n), r(n)]** is the class of languages that can be recognized with by some Turing machine with soundness s (or less) and completeness c (or more) using O(r(n)) random bits and O(q(n)) queries to a proof.

- By definition, NP = $PCP_{1,\, 0}$ [poly(n), 0]

An example

- Graph isomorphism (GI) in NP not known to be in P, nor NP-complete.
- Easy to prove that G and G’ are isomorphic: reveal a permutation of their vertices transforming G to G’.
- Harder to prove that G and G’ are not isomorphic: Write an exponentially long “proof”, listing every permutation of G and G’, and check for duplicates.
- Alternatively, if G and G’ are not isomorphic, write an even longer “proof”: For each N vertex graph, write whether it is isomorphic to G, G’ or neither.

Example Continued

- Second proof can be checked quickly w.h.p.
- STEP 1: Randomly choose G or G’.
  - STEP 2: Randomly select one of the N! possible permutations of the graph’s vertices.
  - STEP 3: Check if the resultant graph, G’’ is listed in the proof as a permutation of G or G’
- If G and G’ are not isomorphic, a proof exists causing our protocol to always accept.
- If they are isomorphic, each G’’ is equally likely to result from G or G’. Any proof fails half the time.
- The number of queries is small, but proof size (and hence number of random bits), is too large.

PCP versus NP

- Any language L in $PCP_{c,\, s}$ [poly(n), log(n)] is recognized by some machine M_L that makes O(poly(n)) queries to a proof for each possible sequence of O(log(n)) random bits.
- Given M_L, there exists a nondeterministic Turing machine M^N_L that recognizes L.
  - On input x, M^N_L "guesses" a proof, then simulates M_L on all sequences of random bits
  - If at least c fraction of sequences accept, x is in L.
- $PCP_{c,\, s}$ [poly(n), log(n)] $\subseteq$ NP

The Power of Randomness

- We just saw $PCP_{c,\,s}[poly(n), log(n)] \subseteq NP$
  - So $PCP_{c,\,s}[poly(n), log(n)] = PCP_{1,\,0}[poly(n), 1]$
- The power of $PCP_{c,\,s}[log(n), poly(n)]$ is not nearly as clear (Solves at least coGI).
- What about when proof are polynomial in length?
  - $PCP_{c,\,s}[log(n), log(n)]$? $PCP_{c,\,s}[1, log(n)]$?

The PCP Theorem

- It turns out NP $\subseteq$ $PCP_{1,\ 1/2}$ [1, log(n)]!
- PCP Theorem (Arora, Lund, Motwani, Sudan, and Szegedy): NP = $PCP_{1,\ 1/2}$ [1, log(n)].
- Recently, a simpler proof was given by Dinur.
  - An NP-complete problem is reduced to a problem in $PCP_{1,\ 1/2}$ [1, log(n)]
- The theorem gives us our first hard to approximate problem.

Why $PCP_{1,\,1/2}\,[1, \log(n)]$

- If $NP = PCP_{1,\,1/2}\,[1, \log(n)]$, then every language in NP can be recognized by a machine that makes a constant number of random queries to a polynomial-sized proof.
- In the spirit of Cook's Theorem, the behavior of these machines can be captured as an instance of SAT.
- Now instances of SAT will have a gap.

An Inapproximability Result

- The following language, PROB, is NP-complete:
  - Let <M, x, 1^t> be a triple consisting of a Turing machine with access to log(t) random bits, a binary input x, and a string of t 1's.
  - <M, x, 1^t> is in the language if M accepts some input <x, y> in t steps with probability p = 1.
  - If M ignores its random bits, PROB is the same as ACCEPT
- Since PROB is NP-complete, any language in NP can be reduced to PROB through some polynomial time reduction, R.
- The PCP Theorem implies R exists such that:
  - M's behavior on <x, y>, when given a particular sequence of random bits, is only a function of O(1) bits of y.
  - OPT = p_{max} cannot be approximated to within a factor of 2.
- The PCP Theorem gives us a gap introducing reduction!

Conclusion

- NP-hard decision problems can be recast as NP-hard optimization problems.
- Often optimization problems are easier to approximate than to solve exactly.
- PCPs allow us to recast NP, using randomness and selectively queried proofs.
- The PCP theorem implies that the NP-complete problem, PROB, does not have a PTAS. Next we:
  - Give a gap preserving reduction from PROB to SAT
  - Give a gap preserving reduction from SAT to 3SAT. As is often the case, the standard reduction already works!