BCCS: Computational methods for complex systems Wednesday, 16 June 2010

Lecture 3

Lecturer: Ashley Montanaro[1] *Hardness of approximation*

1 Overview

In my lectures so far, we've mostly restricted ourselves to *decision* problems, i.e. problems with a yes-no answer. Many problems we wish to solve in practice have a different flavour and can be understood as *optimisation* problems. For example:

- KNAPSACK: given a set of *values* $\{v_i\}$ and *weights* $\{w_i\}$, and a maximum weight M, maximise $\sum_{i=1}^n v_i x_i$, subject to $\sum_{i=1}^n w_i x_i \leq M$, $x_i \in \{0,1\}$.
- TRAVELLING SALESMAN: given a graph G whose edges are weighted with costs, find a tour (cycle) that visits all vertices exactly once and achieves the lowest cost.
- MAX-3SAT: given a CNF boolean formula (not necessarily satisfiable) where each clause contains at most 3 variables, find an assignment to the variables that satisfies the largest number of clauses.
- MAX-E3SAT: the same as MAX-3SAT, but where each clause contains *exactly* 3 variables.

The decision variants of all of these are NP-complete, so we don't expect to be able to solve them exactly in polynomial time. But often in practice we might be satisfied with a "reasonably" approximate answer. The concept of the quality of an approximation can be formalised as follows.

Every optimisation problem P has a set of *feasible solutions* $F(x)$ for each input x, and each solution $s \in F(x)$ has a *value* $v(s)$[2]. If P is a *maximisation* problem, the optimum value is then defined as

$$\mathrm{OPT}(x) = \max_{s \in F(x)} v(s);$$

and if P is a *minimisation* problem,

$$\mathrm{OPT}(x) = \min_{s \in F(x)} v(s).$$

Let A be an algorithm which, given an instance x, returns a feasible solution $A(x) \in F(x)$. If P is a maximisation problem, we say that A is an α-approximation algorithm if, for all x, we have

$$A(x) \geq \alpha\, \mathrm{OPT}(x),$$

and if P is a minimisation problem, we say that A is an α-approximation algorithm if, for all x, we have

$$A(x) \leq \alpha\, \mathrm{OPT}(x).$$

[1] http://www.cs.bris.ac.uk/~montanar/bccs/

[2] This terminology makes sense if we want to maximise the value; for minimisation problems the term "cost" might be more appropriate.

For a maximisation (resp. minimisation) problem P, the *approximation threshold* of P is the largest (resp. smallest) α such that there exists a polynomial-time α-approximation algorithm for P. Beware that there is no standardised terminology in this field; for example, some authors redefine $\alpha \mapsto 1/\alpha$ for maximisation problems, so it is always at least 1.

Intuitively, the approximation threshold of a problem is how well we can expect to solve it in practice. It will turn out that different NP-complete problems can have very different approximation thresholds.

2 The good

Theorem 1. *There is a polynomial-time $(1-\epsilon)$-approximation algorithm for* Knapsack*, for any $\epsilon > 0$.*

Proof. We first use dynamic programming to get a pretty good algorithm for solving Knapsack. Let $V = \max\{v_1, \ldots, v_n\}$ be the maximum value of an item. Define a $(n+1) \times (nV+1)$ table W by letting $W(i, v)$ be the minimum weight attainable by selecting some subset of the first i items, such that their total value is exactly v. $W(0, v) = \infty$ for all v, and then

$$W(i+1, v) = \min\{W(i, v), W(i, v - v_{i+1}) + w_{i+1}\}.$$

Once we've filled in the table, we pick the largest v such that $W(n, v) \leq M$. This algorithm solves Knapsack in time $O(n^2 V)$. Note that this is *not* polynomial in the input size, as V might be very large.

We can make this algorithm run faster, at the cost of introducing inaccuracy, by attempting to reduce V. To do this, we simply truncate the last b bits of each value v_i (for some b to be determined) and replace them with zeroes, which can then be ignored. That is, we get new values $v'_i = 2^b \lfloor v_i / 2^b \rfloor$. We end up with an algorithm that runs in $O(\frac{n^2 V}{2^b})$ time. How bad is the solution we get? Defining S and S' to be the original and new solutions (i.e. sets of items), we have (exercise!)

$$\sum_{i \in S} v_i \geq \sum_{i \in S'} v_i \geq \sum_{i \in S'} v'_i \geq \sum_{i \in S} v'_i \geq \sum_{i \in S} (v_i - 2^b) \geq \sum_{i \in S} v_i - n\, 2^b.$$

Therefore, as V is a lower bound on the value of the optimal solution, this algorithm runs in time $O(\frac{n^2 V}{2^b})$ and outputs a solution that's at most a $(1-\epsilon)$ fraction lower than optimum, where $\epsilon = \frac{n 2^b}{V}$.

But this implies that, for any ϵ, if we choose b such that $2^b = \frac{V\epsilon}{n}$, we obtain an algorithm running in $O(\frac{n^3}{\epsilon})$, which is polynomial in n for any fixed ϵ. □

3 The bad

Theorem 2. *Unless* P = NP*, there is no polynomial-time $(1+\epsilon)$-approximation algorithm for* Travelling Salesman*, for any $\epsilon > 0$.*

Proof. For a contradiction, suppose there is such an algorithm, and call it $\mathcal{A}$. We will use it to construct a polynomial-time algorithm for the NP-complete Hamiltonian Cycle problem.

Given a graph G with n vertices, the algorithm for Hamiltonian Cycle constructs a Travelling Salesman instance with n vertices, where the distance between vertices i and j is 1 if there is an edge in G between i and j, and $(1+\epsilon)n$ otherwise. We now apply our approximation algorithm $\mathcal{A}$ to this problem.

If $\mathcal{A}$ returns a tour of cost exactly n, then we know that G has a Hamiltonian cycle. On the other hand, if $\mathcal{A}$ returns a tour with one or more edges of cost $(1+\epsilon)n$, then the total length of the tour is strictly greater than $(1+\epsilon)n$. As we have assumed that $\mathcal{A}$ never returns a tour with cost greater than $(1+\epsilon)$ times the optimum cost, this means that there is no tour with cost n or less. Therefore, G does not have a Hamiltonian cycle. □

4 The ugly

Theorem 3. *Unless* $\mathsf{P} = \mathsf{NP}$*, there is no polynomial-time* $(\frac{7}{8}+\epsilon)$*-approximation algorithm for* Max-E3Sat*, for any* $\epsilon > 0$*.*

Note that there is an easy randomised $\frac{7}{8}$-approximation algorithm for this problem: just pick an assignment to the variables at random[3]. A random assignment will satisfy each clause with probability $\frac{7}{8}$. By linearity of expectation, this implies that if there are m clauses, on average $\frac{7}{8}m$ clauses will be satisfied. Theorem 3 says that you can't do any better than this trivial algorithm, unless $\mathsf{P} = \mathsf{NP}$.

We will sketch a proof of a weaker variant of this result, which can be stated as follows.

Theorem 4. *Unless* $\mathsf{P} = \mathsf{NP}$*, there is an* $\epsilon > 0$ *such that there is no polynomial-time* $(1-\epsilon)$*-approximation algorithm for* Max-3Sat*.*

Note that we have lost the tight bound on the approximation ratio and have also generalised from Max-E3Sat to Max-3Sat. The proof of Theorem 4 depends on a recently developed, and apparently unrelated, concept in theoretical computer science: probabilistically checkable proofs (PCPs).

4.1 Probabilistically checkable proofs

Imagine we have a proof system where a prover wishes to convince a verifier that the verifier's input is in a language $\mathcal{L}$. Given an n-bit input x and a poly(n)-bit proof, the verifier first looks at x and performs polynomial-time computation on x. The verifier then chooses, at random, a *constant* number of bits of the proof to read. The verifier reads these bits and performs some test on them, either accepting or rejecting the input depending on the result of the test.

The **PCP Theorem** states that any language $\mathcal{L} \in \mathsf{NP}$ has a proof system of this form where:

- For all $x \in \mathcal{L}$, there exists a proof such that the verifier accepts with probability 1.
- For all $x \notin \mathcal{L}$, for any proof, the verifier accepts with probability at most $1/2$.

[3] There is in fact also a $\frac{7}{8}$-approximation algorithm for the more general Max-3Sat problem; this algorithm is based on semidefinite programming and its proof of correctness is considerably more complicated.

In comparison with the usual characterisation of NP, we are looking at much fewer bits of the proof, but at the expense of having some probability of incorrectly identifying strings as being in $\mathcal{L}$ that are actually not in $\mathcal{L}$.

4.2 The Max-qCSP problem

The proof of the PCP Theorem is technical, and beyond the scope of this course. We'll content ourselves with proving that it implies an inapproximability result for Max-3sat. To do this, we first introduce a more general problem: Max-qCSP. A qCSP ("q-ary constraint satisfaction problem") instance is a collection of boolean functions $\phi_i : \{0,1\}^n \to \{0,1\}$ (called constraints), each of which depends on at most q of its input bits. The Max-qCSP problem is to find an input x that satisfies the largest possible number of these constraints. Max-3sat is the special case of Max-3CSP where the constraints are only allowed to be OR functions.

We will show that the PCP Theorem implies inapproximability of Max-qCSP. Consider a language $\mathcal{L} \in \mathsf{NP}$. Given an n-bit input x, using the proof whose existence is guaranteed by the PCP Theorem, we will construct an instance I of Max-qCSP with $m = \text{poly}(n)$ constraints, such that:

- If $x \in \mathcal{L}$, I is satisfiable;
- If $x \notin \mathcal{L}$, the largest number of constraints of I that can be satisfied is $\frac{m}{2}$.

Consider a claimed proof that $x \in \mathcal{L}$, and introduce $\text{poly}(n)$ variables $\{y_i\}$, with one variable corresponding to each bit of the proof. Let R be a string of random bits generated by the verifier. Then R corresponds to a set of bits of the proof to read, and a test function f_R to perform on them. For each R, we produce a constraint corresponding to f_R. This constraint depends on the variables that are read when random string R is generated, and is satisfied if and only if f_R passes. As the verifier only looks at a constant number of bits of the proof, there are at most $O(\log n)$ random bits used, so at most $\text{poly}(n)$ constraints will be produced.

Now, if $x \in \mathcal{L}$, by the PCP Theorem all the constraints are satisfied. On the other hand, if $x \notin \mathcal{L}$, at least half of the constraints must not be satisfied (or the probability of the verifier accepting would be greater than $\frac{1}{2}$).

But this reduction implies that, if we could distinguish between satisfiable instances of Max-qCSP and those for which at most half the constraints are satisfiable, we could solve any problem in NP, and hence $\mathsf{P} = \mathsf{NP}$. This in turn means that Max-qCSP cannot be approximated within a ratio of $\frac{1}{2}$, unless $\mathsf{P} = \mathsf{NP}$.

4.3 Back to Max-3sat

The final step in the proof of Theorem 4 is to show that the result of the previous section implies inapproximability of Max-3sat, which we will achieve by converting the Max-qCSP instance I into a Max-3sat instance I'. First, we rewrite each of the clauses in I as a q-ary CNF expression of size at most 2^q. Second, each of the clauses in this CNF expression can be rewritten as the AND of at most q clauses of length at most 3, using the standard reduction technique from SAT to 3-SAT (which you have found yourself in an exercise). We therefore end up with a CNF expression I' with at most $m' = q\,2^q\,m$ clauses, each of which is of length at most 3.

Now it is clear that, if I is satisfiable, I' is also satisfiable. On the other hand, if an ϵ fraction of the constraints of I are not satisfied, then at least an $\frac{\epsilon}{q\,2^q}$ fraction of the clauses in I' are not satisfied (as each constraint in I corresponds to at most $q\,2^q$ clauses in I'). In particular, we see that the ability to find an approximate solution to Max-3Sat that is at least a $(1 - \frac{1}{q\,2^{q+1}})$ fraction of the optimum implies the ability to find an approximate solution to Max-qCSP that is at least a $\frac{1}{2}$ fraction of the optimum. Taking $\epsilon = \frac{1}{q\,2^{q+1}}$, we have proven Theorem 4: there is a constant $\epsilon > 0$ such that there is no polynomial-time $(1-\epsilon)$-approximation algorithm for Max-3Sat, unless $\mathsf{P} = \mathsf{NP}$.

5 Further reading

The section of this lecture on PCPs is largely based on the Berkeley course "CS294: PCP and Hardness of Approximation" by Luca Trevisan[4]. The Arora-Barak book has many more details about PCPs (Chapters 11 and 22), including the proof of the tight bound for Max-E3Sat. The original proof of the PCP Theorem was the culmination of a series of results drawing together many different concepts in theoretical computer science. Ryan O'Donnell has written a nice history of this[5]. An algorithm for approximating Max-3Sat based on linear programming, which achieves an approximation ratio of $\frac{3}{4}$, is discussed in the Advanced Algorithms course here at Bristol[6].

[4] http://www.cs.berkeley.edu/~luca/pcp/

[5] http://www.cs.washington.edu/education/courses/533/05au/pcp-history.pdf

[6] http://www.maths.bris.ac.uk/~csawh/teaching/aa/lect08.pdf