A brief note on FPTAS and strong NP-hardness

Magnus Lie Hetland

March 30, 2012

It can be shown formally that very few NP-hard problems admit an FPTAS, relating the concept to the class of *strongly* NP-hard problems.[1] An algorithm is *pseudopolynomial* if its running time is polynomial when all numbers in the problem instances are *written in unary*. A problem is *strongly NP-hard problem* if every problem in NP can be reduced to it in polynomial time so that the numbers in the reduced instances are written in unary. This means that all numbers used must be bounded polynomially, and so the problem cannot have a pseudopolynomial algorithm, unless P=NP. It turns out that most problems commonly studied wrt. approximation algorithms are strongly NP-hard. Also, with only some weak restrictions, a problem with an FPTAS has a pseudopolynomial solution! In other words, unless P=NP, no strongly NP-hard problem has an FPTAS.

The extra requirements are that (*i*) the objective function is integer-valued on instances of the problem, and (*ii*) the optimum is polynomial as a function of unary problem size (that is, the size of the instance where every number is written in unary). In this case, if the problem has an FPTAS, it also has a pseudopolynomial solution.

The argument is quite simple. Let's say the optimum value C^* is less than $p(n_u)$, where p is some polynomial, and n_u is the unary problem size. Let the running time of the FPTAS be $q(n, 1/\varepsilon)$, where q is some other polynomial. We set $\varepsilon = 1/p(n_u)$. The result of the FPTAS will (assuming maximization) have a value C less than or equal to

$$(1+\varepsilon) \cdot C^* < C^* + \varepsilon \cdot p(n_u) = C^* + 1\,.$$

In other words, the FPTAS has been *forced to produce the optimum*! The running time is $q(n, p(n_u))$, which is polynomial in n_u, i.e., pseudopolynomial. Furthermore, if $C^* < p(n)$, we have a *polynomial* algorithm. This, however, is less applicable (that is, it holds for fewer problems).

[1] See also § 8.3 in *Approximation Algorithms* by V. Vazirani.