Connecting the Complexities of Approximating Clique and TSP

Richard Chang[†]

Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County

December 8, 2000 11:55

Abstract

This paper demonstrates a connection between the complexity of finding approximate solutions to the MaxClique problem and that of the Traveling Salesman Problem (TSP). The main result of the paper is:

$$\text{MaxClique} \leq_{\text{m}}^{\text{P}} \text{2-approximating MaxClique} \Longrightarrow \text{TSP} \leq_{\text{m}}^{\text{P}} \text{2-approximating TSP},$$

where $\leq_{\text{m}}^{\text{P}}$ denotes many-one reductions between functions. Previously, it was known that TSP $\leq_{\text{m}}^{\text{P}}$-reduces to $(1+n^{-\log n})$-approximating TSP under the same assumption. The proof of the main result uses a characterization of the complexity of NP-approximation problems by nondeterministic bounded query classes. The class $\text{NPF}_b^{\text{SAT}[q(n)]}$ is the class of multi-valued functions computed by NP machines that have access to the NP-complete language SAT as an oracle. These NP machines are limited to $q(n)$ queries to the SAT oracle in the entire nondeterministic computation tree (not just a single computation path). Since finding approximate solutions to MaxClique and TSP are complete for $\text{NPF}_b^{\text{SAT}[q(n)]}$ classes, proving the main result is equivalent to showing that

$$\text{NPF}_b^{\text{SAT}[O(\log n)]} = \text{NPF}_b^{\text{SAT}[\log\log n + O(1)]} \Longrightarrow \text{NPF}_b^{\text{SAT}[n^{O(1)}]} = \text{NPF}_b^{\text{SAT}[O(\log n)]}.$$

This improves upon the previously known result that

$$\text{NPF}_b^{\text{SAT}[O(\log n)]} = \text{NPF}_b^{\text{SAT}[\log\log n + O(1)]} \Longrightarrow \text{NPF}_b^{\text{SAT}[n^{O(1)}]} = \text{NPF}_b^{\text{SAT}[O(\log^2 n)]}.$$

[†]Address: Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250, USA. Email: chang@umbc.edu. Supported in part by National Science Foundation research grant CCR-9610457 and by the University of Maryland Institute for Advanced Computer Studies.

1 Introduction

We investigate the internal complexity of NP-approximation problems and how the internal complexities of two such problems are connected. An example of internal complexity is the self-improvement property of the MaxClique problem. For example, if you are given an "oracle" that finds the vertices of a 4-approximate clique, you can use that algorithm to find 2-approximate cliques by running that same algorithm on the "square" of the input graph. This self-improvement property for MaxClique has been known for a long time [GJ79] and another way to state this property is to say that 4-approximating MaxClique $\leq_{\mathrm{m}}^{\mathrm{P}}$-reduces to 2-approximating MaxClique.[1] While the construction of this self-improvement reduction is fairly straightforward, one would not suppose that solving MaxClique exactly could $\leq_{\mathrm{m}}^{\mathrm{P}}$-reduce to 2-approximating MaxClique or that 2-approximating MaxClique could reduce to $(\log n)$-approximating MaxClique. In fact, if such reductions existed then the Polynomial Hierarchy (PH) would collapse [CKST95].

In this paper, we consider the consequences to the complexity of other NP-approximation problems under the assumption that MaxClique does $\leq_{\mathrm{m}}^{\mathrm{P}}$-reduce to 2-approximating MaxClique. For problems that are closely related to MaxClique, we would expect that a similar reduction can be constructed through the use of approximation-preserving reductions. For example, it is not too surprising to find that Graph Coloring reduces to 2-approximating Graph Coloring under this assumption. On the other hand, the Traveling Salesman Problem[2] (TSP) has a very different complexity from that of MaxClique. Nevertheless, we show that

$$\text{MaxClique} \leq_{\mathrm{m}}^{\mathrm{P}} \text{2-approximating MaxClique} \Longrightarrow \text{TSP} \leq_{\mathrm{m}}^{\mathrm{P}} \text{2-approximating TSP}.$$

Previously, the best known result was that TSP $\leq_{\mathrm{m}}^{\mathrm{P}}$-reduces to $(1 + n^{-\log n})$-approximating TSP under the same assumption [Cha97].

The proof of our main result uses a machine model to characterize the complexity of NP-approximation problems with fixed approximation bounds. To see that this is needed, consider a reduction from the decision problem for TSP to the decision problem for MaxClique. We know such a reduction exists because the decision problem for MaxClique is an NP-complete language. However, if we were asked to construct such a reduction, we would probably resort to Cook's reduction on an NP machine that recognizes the TSP language, and then reduce 3SAT to the MaxClique language.

We have a similar situation here, except we need to extend the NP machine model beyond language recognition. In particular, we need machines that output *solutions* for an NP-approximation problem and guarantee that the solutions are within the specified approximation bound. For example, given a graph that is an instance of TSP, a machine that 2-approximates TSP must output an ordering of the vertices in the graph (i.e., a TSP tour) and guarantee that the tour is not more than twice as long as the optimum tour. An NP machine is not sufficient for this task, because an NP machine doesn't know when the solution it has guessed is optimum or even close enough to optimum.[3] In order to guarantee the quality of its output, the NP machine needs assistance from an oracle.

An $\mathrm{NPF}_b^{\mathrm{SAT}[q(n)]}$ machine is an NP machine that has access to the NP-complete language SAT as an oracle. The machine is limited to $q(n)$ queries to the SAT oracle in the entire nondeterministic computation tree (not just a single computation path). In previous work [Cha97] we have shown that the model is robust (in the sense that there are several equivalent definitions). Furthermore, there are natural complete problems for the classes of multi-valued functions computed by

[1] Note that the $\leq_{\mathrm{m}}^{\mathrm{P}}$-reductions used here are many-one reductions between functions.

[2] We consider only the non-Euclidean version of TSP in this paper.

[3] Given a graph G and a TSP tour t, it is coNP-complete to determine whether the tour is optimum.

$\mathrm{NPF}_b^{\mathrm{SAT}[q(n)]}$ machines — namely, NP-approximation problems with fixed approximation bounds. The following are some examples of known completeness results for the $\mathrm{NPF}_b^{\mathrm{SAT}[q(n)]}$ classes:

1. Finding the optimum solution to TSP is $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for $\mathrm{NPF}_b^{\mathrm{SAT}[n^{O(1)}]}$.
2. Finding a 2-approximate solution to TSP is $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for $\mathrm{NPF}_b^{\mathrm{SAT}[O(\log n)]}$.
3. Finding the optimum solution to MaxClique is $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for $\mathrm{NPF}_b^{\mathrm{SAT}[O(\log n)]}$.
4. Finding a 2-approximate solution to MaxClique is $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for $\mathrm{NPF}_b^{\mathrm{SAT}[\log\log n+O(1)]}$.

It is fairly easy to see that statements (1) and (3) above hold. Krentel showed that $n^{O(1)}$ queries to SAT are sufficient to find the *length* of the optimum TSP tour [Kre88]. Once the length is known, an NP machine can guess a tour with that length. Completeness follows from a straightforward construction. Similarly, $O(\log n)$ queries are sufficient to find the *size* of largest clique in a graph and then nondeterministic guessing can be used to find a clique with the optimum size. On the other hand, proving that 2-approximating MaxClique is complete for $\mathrm{NPF}_b^{\mathrm{SAT}[\log\log n+O(1)]}$ required the use of the witness-preserving version of the PCP proof of the non-approximability MaxClique [FGL$^+$96, AS98, ALM$^+$98, Aro94]. The completeness results for 2-approximating TSP were proven by direct construction.

Given these completeness results, proving the main result becomes equivalent to showing that

$$\mathrm{NPF}_b^{\mathrm{SAT}[O(\log n)]} \subseteq \mathrm{NPF}_b^{\mathrm{SAT}[\log\log n+O(1)]} \Longrightarrow \mathrm{NPF}_b^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{NPF}_b^{\mathrm{SAT}[O(\log n)]}.$$

In previous work, we were only able to show that $\mathrm{NPF}_b^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{NPF}_b^{\mathrm{SAT}[O(\log^2 n)]}$ using a rather involved hard/easy argument. In this paper we use a different technique to show that

$$\mathrm{NPF}_b^{\mathrm{SAT}[O(\log n)]} = \mathrm{NPF}_b^{\mathrm{SAT}[\log\log n+O(1)]} \Longrightarrow \mathrm{NPF}_b^{\mathrm{SAT}[O(\log^2 n)]} \subseteq \mathrm{NPF}_b^{\mathrm{SAT}[\log\log n+O(1)]}.$$

Combined with the previous results, we have the desired main theorem:

$$\text{MaxClique} \leq_{\mathrm{m}}^{\mathrm{P}} \text{2-approximating MaxClique} \Longrightarrow \text{TSP} \leq_{\mathrm{m}}^{\mathrm{P}} \text{2-approximating TSP}.$$

Our main theorem also demonstrates the utility of a machine model in the study of the complexity of NP-approximation problems. The model we propose, the $\mathrm{NPF}_b^{\mathrm{SAT}[q(n)]}$ classes, might not be the cleanest model, but all of its components appear necessary. For example, to 2-approximate MaxClique, we use $\log\log n$ queries to determine via binary search which of the intervals $[1,2),[2,4),[4,8),\ldots$ contains the size of the largest clique. Let that interval be $[m,2m)$. Then, we nondeterministically guess a clique with m vertices. That clique is guaranteed to be a 2-approximate clique. Thus, we used nondeterminism to guess a feasible solution and the oracle queries to SAT to check that the solution is close enough. Given that 2-approximating MaxClique is $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for the class of functions computed by $\mathrm{NPF}_b^{\mathrm{SAT}[\log\log n+O(1)]}$ machines, it seems fairly safe to say that any complexity class that characterizes the complexity of 2-approximating MaxClique must also capture the properties of these nondeterministic bounded query classes. So, it would appear that nondeterminism and oracle queries are both sufficient and necessary to produce solutions to these NP-approximation problems.

In fact there is a fairly long history of using bounded query complexity to measure the complexity of NP-optimization problems. Krentel [Kre88] first used deterministic bounded query classes to classify NP-optimization problems. Chang, Gasarch and Lund showed the relationship between the number of queries to SAT and the complexity of approximating the size of the largest clique in

a graph [CG93, CGL97, Cha96]. In general they showed that more oracle queries are needed to find closer approximations. These results also extend to several NP-optimization problems including Graph Coloring. Chang [Cha97] further extended these results to the case where one must produce a witness for the NP-approximation problem — e.g., output the vertices in a 2-approximate clique not just its size. Additional results connecting bounded queries and approximations can be found in the works of Crescenzi, Trevisan, Kann and Silvestri [CT94, CKST95].

2 Definitions and Facts

In this section, we present the definitions and results from prior work that we need to prove the main theorem. First, we formally define the terms "$k(\cdot)$-approximate MaxClique" and "$k(\cdot)$-approximates TSP." In the definition of TSP below, we do not require the weight function to satisfy the triangle inequality. Thus, finding constant factor approximations of this version of TSP is NP-hard. In particular, recent work on approximation schemes for Euclidean TSP [Aro96, Mit99, Aro97] are not applicable.

Definition 1 Let $G = (V, E)$ be an undirected graph with n vertices and let $k(n)$ be an approximation factor such that $\forall n, 1 \leq k(n) \leq n$. We use $\omega(G)$ to denote the size of a largest clique in G. We say that a multi-valued function f $k(\cdot)$-approximates MaxClique if for all graphs G every output of $f(G)$ is a set $X \subseteq V$ such that X is a clique and $|X| \geq \omega(G)/k(n)$.

Definition 2 Let $G = (V, E)$ be a weighted undirected graph with n vertices and weight function $w : E \to \mathbb{N}$. Without loss of generality, we assume that

$$\sum_{e \in E} w(e) \leq 2^n.$$

A TSP tour in the graph is a cycle that visits each vertex exactly once. The length of a TSP tour is sum of the weights of the edges in the cycle. Let OptTSP(G) denote the length of a shortest TSP tour in G and let $k(n)$ be an approximation factor such that $\forall n, 1 \leq k(n) \leq 2^n$. We say that a multi-valued function f $k(\cdot)$-approximates TSP if for all graphs G every output of $f(G)$ is a TSP tour of G with length $\leq k(n)$OptTSP(G).

An approximation problem with a fixed approximation bound can be modeled mathematically as a multi-valued function. A multi-valued function may have several outputs for each input string. It may seem awkward to have to work with multi-valued functions. However, when we ask for a 2-approximate solution for an instance of MaxClique, we are inherently saying that we do not care which of many possible solutions is produced. We use the following generalization of many-one reductions for reductions between multi-valued functions.

Definition 3 Let f and g be two multi-valued functions. We say that $f \leq^{\mathrm{P}}_{\mathrm{m}} g$ if there exist two polynomial-time computable functions T_1 and T_2 such that for every input x of f, $T_1(x) = y$ is a string in the domain of g and for every output of z of $g(y)$, $T_2(x, z)$ is an output of $f(x)$.

For example, Graph Coloring $\leq^{\mathrm{P}}_{\mathrm{m}}$ MaxClique. By this we mean that given any graph G, we can produce a graph G' in polynomial time such that given the vertices of any maximum clique G' (there can be many maximum cliques), we can then produce in polynomial time a coloring of G that uses a minimum number of colors.

In this paper, we use oracle Turing machines with limited access to the SAT oracle. We will start with the definition for deterministic bounded query classes.

Definition 4 Let $q(n)$ be a polynomial-time computable function. We use $\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ to denote the set of functions computed by deterministic polynomial-time Turing machines which ask at most $q(n)$ queries to the SAT oracle on inputs of length n. Functions in $\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ must be single-valued.

Definition 5 Let A be any language. We use $A(x)$ to denote the characteristic function of the set A at x. We define $\chi^A_\omega(x_1, \ldots, x_m)$ to be an m-bit string such that the i^{th} bit is 1 if and only if $x_i \in A$. That is, $\chi^A_\omega(x_1, \ldots, x_m) = A(x_1) \cdots A(x_m)$, where juxtaposition means concatenation.

We will sometimes use a polynomial-time machine (deterministic or nondeterministic) without access to a SAT oracle to simulate a $\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ computation. If M is a $\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ machine and $b \in \{0,1\}^{q(n)}$, then a polynomial-time machine M' given the string b can simulate M on input x as follows. M' on input (x, b) simulates $M(x)$ step by step until M makes an oracle query. Since M' does not have access to an oracle, M' uses the i^{th} bit of b as the answer to the oracle query (1 means the query string is in SAT). If $M(x)$ makes the queries $a_1, \ldots, a_{q(n)}$ and $b = \chi^{\mathrm{SAT}}_\omega(a_1, \ldots, a_{q(n)})$, then the behaviors of $M(x)$ and $M'(x, b)$ are identical. If $b \neq \chi^{\mathrm{SAT}}_\omega(a_1, \ldots, a_{q(n)})$, then the behavior on $M'(x, b)$ is not predictable, but will nevertheless terminate in polynomial time.

Definition 6 Let $q(n)$ be a polynomial-time computable function. We use $\mathrm{NPF}_b^{\mathrm{SAT}[q(n)]}$ to denote the set of *total multi-valued functions* computed by nondeterministic polynomial-time Turing machines which ask at most $q(n)$ queries to the SAT oracle in the entire nondeterministic computation tree on inputs of size n. The class $\mathrm{NP}_b^{\mathrm{SAT}[q(n)]}$ is the analogous class of languages.

A potentially ambiguous point about the definition of $\mathrm{NPF}_b^{\mathrm{SAT}[q(n)]}$ is whether the same query asked on two different computation paths counts as one query or two queries. It turns out that this does not matter for our applications, but for the sake of mathematical rigor we will count this as two queries.

It is not useful to limit the number of oracle queries made by an NP machine on each computation *path* because in that case one query is as powerful as polynomially many queries and $\mathrm{NP}^{\mathrm{SAT}[1]}$ would simply be Σ^{P}_2. Nevertheless, to avoid any potential confusion, we use the subscript b in $\mathrm{NPF}_b^{\mathrm{SAT}[q(n)]}$ to indicate that $q(n)$ is a bound on the number of oracle queries in the entire computation tree (as is done in [BDG90]). Counting queries in this manner was also used by Book, Long and Selman [BLS84, Lon85] and by Wagner [Wag90]. In addition, our restriction of the $\mathrm{NPF}_b^{\mathrm{SAT}[q(n)]}$ classes to total functions is not overly limiting. For example, using Fact 11 one can show that for $q(n) \in O(\log n)$ every partial function computable by a nondeterministic Turing machine using $q(n)$ queries to SAT has a total extension in $\mathrm{NPF}_b^{\mathrm{SAT}[q(n)+1]}$. (On inputs where the original function is undefined, the total extension outputs a new $\perp$ symbol.)

Since a $\leq^{\mathrm{P}}_{\mathrm{m}}$-reduction can stretch the length of its output by a polynomial factor, the class $\mathrm{NPF}_b^{\mathrm{SAT}[O(\log n)]}$ is closed under $\leq^{\mathrm{P}}_{\mathrm{m}}$-reductions whereas the class $\mathrm{NPF}_b^{\mathrm{SAT}[\log n]}$ is not. This polynomial stretching can also be used to in a padding argument to show that for all $c_1 > c_2 > 0$, every function in $\mathrm{NPF}_b^{\mathrm{SAT}[c_1 \log n]}$ reduces to some function f' in $\mathrm{NPF}_b^{\mathrm{SAT}[c_2 \log n]}$. Similar closure and padding properties hold for the classes $\mathrm{NPF}_b^{\mathrm{SAT}[q(n)]}$, where $q(n) = n^{O(1)}$, $q(n) = O(\log^a n)$ and $q(n) = \log \log n + O(1)$.

We will use the following results which show that approximating MaxClique and TSP are $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for various $\mathrm{NPF}_b^{\mathrm{SAT}[q(n)]}$ classes.

Fact 7 [Cha97]

- MaxClique is $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for $\mathrm{NPF}_b^{\mathrm{SAT}[O(\log n)]}$.
- $(1+1/\log^a n)$-approximating MaxClique is $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for $\mathrm{NPF}_b^{\mathrm{SAT}[(a+1)\log\log n+O(1)]}$.
- 2-approximating MaxClique is $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for $\mathrm{NPF}_b^{\mathrm{SAT}[\log\log n+O(1)]}$.
- $(\log n)$-approximating MaxClique is $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for $\mathrm{NPF}_b^{\mathrm{SAT}[\log\log n-\log\log\log n+O(1)]}$.

Fact 8 [Cha97]

- TSP is $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for $\mathrm{NPF}_b^{\mathrm{SAT}[n^{O(1)}]}$.
- $(1+n^{-\log^a n})$-approximating TSP is $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for $\mathrm{NPF}_b^{\mathrm{SAT}[O(\log^{a+1} n)]}$.
- $(1+n^{-\log n})$-approximating TSP is $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for $\mathrm{NPF}_b^{\mathrm{SAT}[O(\log^2 n)]}$.
- For constant k, k-approximating TSP is $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for $\mathrm{NPF}_b^{\mathrm{SAT}[O(\log n)]}$.

It turns out that $\mathrm{NPF}_b^{\mathrm{SAT}[q(n)]}$ computations can be put into a very convenient normal form where all the queries are made in an initial deterministic phase. We will make use of the following facts in the proof of the main theorem.

Definition 9 Let $\mathcal{C}$ be a class of functions. Then, $\mathrm{NPF}//\mathcal{C}$ is the set of total multi-valued functions f defined by a function $g \in \mathcal{C}$ and an NP machine N such that the outputs of $f(x)$ are the outputs of $N(x, g(x))$.

Fact 10 [Cha97] Let $r(n) \in n^{O(1)}$ be a polynomial-time computable function, then

$$\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[r(n)]} \subseteq \mathrm{NPF}_b^{\mathrm{SAT}[r(n)]} \subseteq \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[2r(n)]}.$$

Fact 11 [Cha97] Let $r(n) \in O(\log n)$ be a polynomial-time computable function, then

$$\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[r(n)]} = \mathrm{NPF}_b^{\mathrm{SAT}[r(n)]}.$$

Fact 12 [Cha97]

$$\mathrm{NPF}_b^{\mathrm{SAT}[O(\log n)]} \subseteq \mathrm{NPF}_b^{\mathrm{SAT}[\log\log n+O(1)]} \Longrightarrow \mathrm{NPF}_b^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{NPF}_b^{\mathrm{SAT}[O(\log^2 n)]}.$$

The preceding fact is sufficient to show that if MaxClique $\leq_{\mathrm{m}}^{\mathrm{P}}$-reduces to 2-approximating MaxClique, then TSP $\leq_{\mathrm{m}}^{\mathrm{P}}$-reduces to $(1+n^{-\log n})$-approximating TSP. In the next section, we prove our main result which shows that in fact, TSP $\leq_{\mathrm{m}}^{\mathrm{P}}$-reduces to 2-approximating TSP.

3 Main Theorem

The proofs in this section are best described as oracle replacement proofs. Under the assumption that $\mathrm{NPF}_b^{\mathrm{SAT}[O(\log n)]} = \mathrm{NPF}_b^{\mathrm{SAT}[O(\log\log n+O(1)]}$, we can essentially replace $\log n$ queries to SAT with $\log\log n$ queries. However, there are some subtleties and complications that are best illustrated by considering the deterministic case first. Although we do not use the statement of the next lemma in the proof of the main theorem, we will use the proof of the lemma as a template.

Lemma 13

$$\mathrm{PF}^{\mathrm{SAT}[O(\log n)]} = \mathrm{PF}^{\mathrm{SAT}[\log\log n + O(1)]} \Longrightarrow \mathrm{PF}^{\mathrm{SAT}[O(\log^2 n)]} \subseteq \mathrm{PF}^{\mathrm{SAT}[\log\log n + O(1)]}.$$

It is tempting to try to prove Lemma 13 using the following erroneous approach. Let f be a function in $\mathrm{PF}^{\mathrm{SAT}[O(\log^2 n)]}$ via a deterministic polynomial-time machine M. Fix an input x of length n. Let $g(x) = \chi_\omega^{\mathrm{SAT}}(q_1, \ldots, q_r)$ where $r = \log n$ and $q_1, \ldots, q_r$ are the first r queries asked by M on input x. Obviously $g(x) \in \mathrm{PF}^{\mathrm{SAT}[O(\log n)]}$ which is contained in $\mathrm{PF}^{\mathrm{SAT}[\log\log n + O(1)]}$ by hypothesis. Now, we can construct a new machine M' that uses the $\mathrm{PF}^{\mathrm{SAT}[\log\log n + O(1)]}$ machine for $g(x)$ to obtain the answer to the first $\log n$ queries made by M on input x. This process reduces the number of queries used to compute $f(x)$ by about $\log n - \log\log n$. We then repeat the process until our machine uses fewer than $\log n$ queries in total. This would give us a procedure to compute $f(x)$ in $\mathrm{PF}^{\mathrm{SAT}[O(\log n)]}$ which is equal to $\mathrm{PF}^{\mathrm{SAT}[\log\log n]}$.

The error in this approach is that each time we reduce the number of queries, we might increase the length of some queries by a polynomial factor. For example, the $\mathrm{PF}^{\mathrm{SAT}[\log\log n + O(1)]}$ procedure which computes $g(x)$ might ask queries that have lengths that are the squares of the lengths of the queries made by the original $\mathrm{PF}^{\mathrm{SAT}[O(\log n)]}$ machine. Repeated application of this process results in repeated squaring of the lengths of the oracle queries. The result is a super-polynomial running time. We can correct this error by replacing the oracle queries in a uniform manner.

Proof: Let f be a function in $\mathrm{PF}^{\mathrm{SAT}[k\log^2 n]}$ via a deterministic polynomial-time machine M. For $x \in \Sigma^*$ and $b_i \in \{0,1\}$, let $h(x, b_1 \cdots b_m) = \chi_\omega^{\mathrm{SAT}}(q_1, \ldots, q_r)$ where $r = \log|x|$ and $q_1, \ldots, q_r$ are the $(m+1)^{th}$ through the $(m+r)^{th}$ queries asked by $M(x)$ assuming that $b_1 \cdots b_m$ are the answers to the first m queries. We use the function h in the following way. Suppose that we already have the answers to the first m queries asked by $M(x)$ and these answers are $b_1, \ldots, b_m$ which we can represent as a string in $b \in \{0,1\}^m$. Then, $h(x, b)$ computes the answers to the next $\log|x|$ queries made by $M(x)$. Clearly, $h(x, b)$ can be computed in $\mathrm{PF}^{\mathrm{SAT}[\log n]}$. Thus, there is a $\mathrm{PF}^{\mathrm{SAT}[\log\log n + O(1)]}$ machine M_h for h. Hence, we have the following $\mathrm{PF}^{\mathrm{SAT}[O(\log n \log\log n)]}$ algorithm for $f(x)$.

Procedure 1

1. Let x be the input string of length n.
2. Let $b := \epsilon$, $m := 0$ and $r := \log|x|$.
3. Compute $c := h(x, b)$ using M_h and $\log\log n$ queries to SAT.
4. Let $m := m + r$ and $b := bc$
5. If $m < k\log^2|x|$, goto Step 3
6. Simulate $M(x)$ using b as the answers to the oracle queries.

Procedure 1 works by replacing blocks of $\log n$ queries with $\log\log n$ queries. The queries made by M_h may be much longer than the queries made by M. However, the length of the queries are bounded by a single polynomial (namely, the running time of M_h on inputs of length $n + \log^2 n$). We can repeat this process one more time and replace blocks of $\log n$ queries made in Procedure 1 with $\log\log n$ queries. This time, there are only $\log\log n$ blocks, so the total number of queries is $(\log\log n)^2$. Thus, $f \in \mathrm{PF}^{\mathrm{SAT}[(\log\log n)^2]}$. Since $(\log\log n)^2 \in O(\log n)$, it follows that $f \in \mathrm{PF}^{\mathrm{SAT}[O(\log n)]}$ which equals $\mathrm{PF}^{\mathrm{SAT}[\log\log n + O(1)]}$. Thus, $\mathrm{PF}^{\mathrm{SAT}[O(\log^2 n)]} = \mathrm{PF}^{\mathrm{SAT}[\log\log n + O(1)]}$.

□

The proof of the main theorem follows the same outline as the proof of Lemma 13. We will replace blocks of $\log n$ queries with $\log\log n$ queries. We will do this in two phases to reduce the $\log^2 n$ queries used by the original machine to $\log n \log\log n$ queries and then to $(\log\log n)^2$ queries. There is a catch, however. If we copied Procedure 1 as is, each iteration of the loop will contain a query step and a nondeterministic step. Since we are required to count the queries in the entire nondeterministic computation tree, the number of queries would actually increase. To circumvent this difficulty, we modify the procedure so all the queries are made deterministically.

Theorem 14

$$\mathrm{NPF}_b^{\mathrm{SAT}[O(\log n)]} = \mathrm{NPF}_b^{\mathrm{SAT}[\log\log n+O(1)]} \Longrightarrow \mathrm{NPF}_b^{\mathrm{SAT}[O(\log^2 n)]} \subseteq \mathrm{NPF}_b^{\mathrm{SAT}[\log\log n+O(1)]}.$$

Proof: By the normal forms for $\mathrm{NPF}_b^{\mathrm{SAT}[q(n)]}$ computations in Fact 10, for all constants c,

$$\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[c\log^2 n]} \subseteq \mathrm{NPF}_b^{\mathrm{SAT}[c\log^2 n]} \subseteq \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[2c\log^2 n]}.$$

Hence $\mathrm{NPF}_b^{\mathrm{SAT}[O(\log^2 n)]} = \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[O(\log^2 n)]}$. Furthermore, by Fact 11, we have

$$\mathrm{NPF}_b^{\mathrm{SAT}[O(\log n)]} = \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[O(\log n)]}$$

and

$$\mathrm{NPF}_b^{\mathrm{SAT}[\log\log n+O(1)]} = \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[\log\log n+O(1)]}.$$

Thus, it suffices to show that $\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[O(\log^2 n)]} \subseteq \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[O(\log n)]}$.

Let f be a function in $\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[k\log^2 n]}$ via a $\mathrm{PF}^{\mathrm{SAT}[k\log^2 n)]}$ machine M and an NP machine N. We define the function h as in Lemma 13. That is, $h(x, b_1\cdots b_m) = \chi_\omega^{\mathrm{SAT}}(q_1,\ldots,q_r)$ where $r = \log|x|$ and $q_1,\ldots,q_r$ are the $(m+1)^{th}$ through the $(m+r)^{th}$ queries asked by $M(x)$ assuming that $b_1\cdots b_m$ are the answers to the first m queries. Clearly, $h \in \mathrm{PF}^{\mathrm{SAT}[\log n]}$ so by hypothesis there exists an $\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[\log\log n+O(1)]}$ computation for h via a $\mathrm{PF}^{\mathrm{SAT}[\log\log n+O(1)]}$ machine M_h and an NP machine N_h. (We cannot assume here that $h \in \mathrm{PF}^{\mathrm{SAT}[\log\log n+O(1)]}$.)

Now consider the following procedure that computes the output of $M(x)$ and uses the output to compute $f(x)$. We won't run this procedure directly, but will eventually simulate it with an $\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[O(\log n\log\log n)]}$ computation.

Procedure 2

1. Let x be the input string.
2. Let $b := \epsilon$, $m := 0$ and $r := \log|x|$.
3. Compute $z := M_h(x, b)$ using $\log\log n$ queries to SAT.
4. Compute $c := N_h(x, b, z)$.

   (Note that $c = h(x, b)$, $c \in \{0,1\}^r$ and that $N_h(x, b, z)$ is single-valued.)
5. Let $m := m + r$ and $b := bc$
6. If $m < k\log^2|x|$, goto Step 3
7. Simulate $M(x)$ using b as the answers to the oracle queries and obtain the output string w.
8. Simulate $N(x, w)$ to compute the value of $f(x)$.

Procedure 2 computes $f(x)$ correctly, but accounting for the number of queries used is problematic. First, Procedure 2 is definitely not an $\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ computation, because the loop in Steps 3 through 6 interleave oracle queries and nondeterministic computations. (Recall that an $\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[q(n)]}$ computation must ask all the queries in an initial deterministic phase.) We could consider Procedure 2 an $\mathrm{NPF}_b^{\mathrm{SAT}[q(n)]}$ computation where each computation path asks $O(\log n \log\log n)$ queries to SAT. However, in this case we must count the queries in the entire nondeterministic computation tree, not just the number of queries in a single computation path. The problem here is that each time the loop is executed we may have 2 or more computation paths in Step 4 that output the string c. After $O(\log n)$ iterations of the loop we have a polynomial number of paths in the nondeterministic computation tree — each of which makes some oracle queries. Thus, the number of queries by this accounting actually goes up.

We resolve this difficulty by taking advantage of the fact that the output from $N_h(x, b, z)$ is unique. That is, despite the fact that many nondeterministic branches of Procedure 2 may make queries to the oracle, every branch that does so asks the same sequence of queries. Now consider the following language:

$$Q = \{\langle x, d_1 \cdots d_s\rangle \mid \text{there exists a computation path in Procedure 2 where the first } s \text{ queries are answered according to } d_1 \cdots d_s \text{ and the } (s+1)^{th} \text{ query is in SAT.}\}$$

When $d_1 \cdots d_s$ are indeed the answers to the first s queries made in Procedure 2, the $(s+1)^{th}$ query is unique. However, if $d_1, \ldots, d_s$ are not the correct answers, the nondeterministic computation of $N_h(x, b, z)$ does have to guarantee unique output. This merely explains the somewhat convoluted definition of Q. We won't actually need to consider the case where $d_1 \cdots d_s$ are not the correct answer to the oracle queries made in Procedure 2. The important thing is that Q is an NP language, since an NP machine can simulate Procedure 2 if it is given the answers to the oracle queries. The trick now is to ask the SAT oracle questions about membership in Q. This allows us to ask all of the queries in an initial deterministic phase.

Procedure 3

1. Let x be the input string of length n.
2. Let $t = k' \log n \log\log n$ be a bound on the number of oracle queries asked on any single computation path of Procedure 2 for inputs of length n.
3. Let $d := \epsilon$.
4. for $i := 1$ to t, if $\langle x, d\rangle \in Q$ then $d := d1$ else $d := d0$.

   (This loop uses t queries to SAT.)
5. Nondeterministically simulate Procedure 2 using d as the answers to the oracle queries and obtain the output string w.
6. Output w as the value for $f(x)$.

Procedure 3 shows that $f \in \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[O(\log n \log\log n)]}$. Thus, we have shown that

$$\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[O(\log^2 n)]} \subseteq \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[O(\log n \log\log n)]}.$$

Finally, we finish proving the theorem by repeating the entire process one more time and get $f \in \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[O((\log\log n)^2)]}$. Since $(\log\log n)^2 \in O(\log n)$, we have

$$\mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[O(\log^2 n)]} \subseteq \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[O(\log n)]} = \mathrm{NPF}//\mathrm{PF}^{\mathrm{SAT}[\log\log n + O(1)]}. \qquad \square$$

Theorem 15

$$\textsc{MaxClique} \leq_{\mathrm{m}}^{\mathrm{P}} \text{2-approximating } \textsc{MaxClique} \Longrightarrow \mathrm{TSP} \leq_{\mathrm{m}}^{\mathrm{P}} \text{2-approximating TSP}.$$

Proof: By the completeness results for MaxClique in Fact 7,

$$\textsc{MaxClique} \leq_{\mathrm{m}}^{\mathrm{P}} \text{2-approximating } \textsc{MaxClique} \Longrightarrow \mathrm{NPF}_b^{\mathrm{SAT}[O(\log n)]} \subseteq \mathrm{NPF}_b^{\mathrm{SAT}[\log\log n + O(1)]}.$$

Furthermore, by Theorem 14 and Fact 12, we have

$$\mathrm{NPF}_b^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{NPF}_b^{\mathrm{SAT}[O(\log^2 n)]} \subseteq \mathrm{NPF}_b^{\mathrm{SAT}[O(\log n)]} \subseteq \mathrm{NPF}_b^{\mathrm{SAT}[\log\log n + O(1)]}.$$

Finally, by the completeness results for TSP in Fact 8, TSP $\leq_{\mathrm{m}}^{\mathrm{P}}$-reduces to 2-approximating TSP. □

The previous theorem can be stated in a tighter form since the collapse of the $\mathrm{NPF}_b^{\mathrm{SAT}[q(n)]}$ classes really implies that the four NP-approximation problems are all $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for the class $\mathrm{NPF}_b^{\mathrm{SAT}[\log\log n + O(1)]}$ and hence are equivalent under $\leq_{\mathrm{m}}^{\mathrm{P}}$-reductions.

Theorem 16

$$\textsc{MaxClique} \leq_{\mathrm{m}}^{\mathrm{P}} \text{2-approximating } \textsc{MaxClique}$$

$$\Longrightarrow \mathrm{TSP} \equiv_{\mathrm{m}}^{\mathrm{P}} \text{2-approximating TSP} \equiv_{\mathrm{m}}^{\mathrm{P}} \textsc{MaxClique} \equiv_{\mathrm{m}}^{\mathrm{P}} \text{2-approximating } \textsc{MaxClique}.$$

4 Conclusion

Our main result is essentially an upward collapse result for nondeterministic bounded query classes. Because NP-approximation problems are complete functions for these classes, we also have the analogous results about NP-approximation problems. In the preceding proofs, we take full advantage of the fact that we have a collapse from $O(\log n)$ queries down to $\log\log n + O(1)$ queries. Previous attempts to prove the main theorem failed, because they used the hard/easy argument which only uses the collapse of a single level of the nondeterministic query hierarchy — i.e., that $\mathrm{NPF}_b^{\mathrm{SAT}[f(n)+1]} = \mathrm{NPF}_b^{\mathrm{SAT}[f(n)]}$ for some $f(n) \in \log\log n + O(1)$. In fact, it still remains open whether for $f(n) \in \log\log n + O(1)$,

$$\mathrm{NPF}_b^{\mathrm{SAT}[f(n)+1]} \subseteq \mathrm{NPF}_b^{\mathrm{SAT}[f(n)]} \Longrightarrow \mathrm{NPF}_b^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{NPF}_b^{\mathrm{SAT}[f(n)]}.$$

The best result is still that

$$\mathrm{NPF}_b^{\mathrm{SAT}[f(n)+1]} \subseteq \mathrm{NPF}_b^{\mathrm{SAT}[f(n)]} \Longrightarrow \mathrm{NPF}_b^{\mathrm{SAT}[n^{O(1)}]} \subseteq \mathrm{NPF}_b^{\mathrm{SAT}[O(\log^2 n)]}.$$

As for other open problems, we still have no adverse consequences (e.g., that PH collapses) to the assumption that $\mathrm{NPF}_b^{\mathrm{SAT}[n^{O(1)}]} = \mathrm{NPF}_b^{\mathrm{SAT}[O(\log n)]}$. In terms of NP-approximation problems, it remains entirely possible, though we would conjecture otherwise, that TSP does reduce to 2-approximating TSP.

References

[ALM+98] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, May 1998.

[Aro94] S. Arora. *Probabilistic Checking of Proofs and Hardness of Approximation Problems*. PhD thesis, University of California Berkeley, August 1994. Revised version available as Technical Report CS-TR-476-94.

[Aro96] S. Arora. Polynomial time approximation schemes for Euclidean TSP and other geometric problems. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 2–11, 1996.

[Aro97] S. Arora. Nearly linear time approximation schemes for Euclidean TSP and other geometric problems. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 554–563, 1997.

[AS98] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70–122, January 1998.

[BDG90] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity II*, volume 22 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1990.

[BLS84] R. V. Book, T. J. Long, and A. L. Selman. Quantitative relativizations of complexity classes. *SIAM Journal on Computing*, 13:461–487, 1984.

[CG93] R. Chang and W. I. Gasarch. On bounded queries and approximation. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 547–556, November 1993.

[CGL97] R. Chang, W. I. Gasarch, and C. Lund. On bounded queries and approximation. *SIAM Journal on Computing*, 26(1):188–209, February 1997.

[Cha96] R. Chang. On the query complexity of clique size and maximum satisfiability. *Journal of Computer and System Sciences*, 53(2):298–313, October 1996.

[Cha97] R. Chang. Bounded queries, approximations and the Boolean hierarchy. Technical Report TR 97-035, Electronic Colloquium on Computational Complexity, 1997. To appear in *Information and Computation*.

[CKST95] P. Crescenzi, V. Kann, R. Silvestri, and L. Trevisan. Structure in approximation classes. In *Proceedings of the 1st Computing and Combinatorics Conference*, volume 959 of *Lecture Notes in Computer Science*, pages 539–548. Springer-Verlag, August 1995.

[CT94] P. Crescenzi and L. Trevisan. On approximation scheme preserving reducibility and its applications. In *Proceedings of the 14th Conference on the Foundations of Software Technology and Theoretical Computer Science*, volume 880 of *Lecture Notes in Computer Science*, pages 330–341. Springer-Verlag, December 1994.

[FGL+96] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating clique. *Journal of the ACM*, 43(2):268–292, March 1996.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[Kre88] M. W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36(3):490–509, 1988.

[Lon85] T. J. Long. On restricting the size of oracles compared with restricting access to oracles. *SIAM Journal on Computing*, 14:585–597, 1985.

[Mit99] J. S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM Journal on Computing*, 28(4):1298–1309, August 1999.

[Wag90] K. Wagner. Bounded query classes. *SIAM Journal on Computing*, 19:833–846, 1990.