Constant-Time Approximation Algorithms via Local Improvements

Huy N. Nguyen
MIT, CSAIL
huy2n@mit.edu

Krzysztof Onak*
MIT, CSAIL
konak@mit.edu

Abstract

We present a technique for transforming classical approximation algorithms into constant-time algorithms that approximate the size of the optimal solution. Our technique is applicable to a certain subclass of algorithms that compute a solution in a constant number of phases. The technique is based on greedily considering local improvements in random order.

The problems amenable to our technique include Vertex Cover, Maximum Matching, Maximum Weight Matching, Set Cover, and Minimum Dominating Set. For example, for Maximum Matching, we give the first constant-time algorithm that for the class of graphs of degree bounded by d, computes the maximum matching size to within εn, for any $\varepsilon > 0$, where n is the number of nodes in the graph. The running time of the algorithm is independent of n, and only depends on d and ε.

1. Introduction

There has been an enormous amount of work on Maximum Matching, Vertex Cover, and Set Cover in the classical computation model, where the whole input is read. It is obviously not possible to compute a solution to them in time sublinear in input size, since an optimal solution may itself have linear size. Can one approximate just the optimal solution size in time sublinear in the input size? This and similar questions have been asked by several researchers for various optimization problems [1, 2, 3, 9, 14]. In particular, Parnas and Ron [14] asked this question for the minimum vertex cover problem. They discovered a connection to distributed algorithms that run in a constant number of rounds. For graphs of bounded maximum or average degree, the connection yields approximation algorithms of running time independent of the size of the graph. In this paper, we show a general technique that can be used to construct constant-time approximation algorithms. The technique works for all problems that were considered by Parnas and Ron, but does not rely on distributed algorithms, and for Maximum Matching, it can be used to overcome limitations of the previously known distributed algorithms.

*Supported in part by NSF grant 0514771.

1.1. Simple Example: Maximal Matching

We begin with the example of approximating the size of a maximal matching in a graph of maximum degree at most d. We consider a run of our algorithm to be *successful* if the returned value approximates the size of *some* maximal matching.

Algorithm. All of our algorithms follow a general framework of Parnas and Ron [14], which first shows that the computational problem can be easily solved assuming oracle access to a solution, and then shows how to implement the oracle.

In this case, let M be a fixed maximal matching. Suppose that we have access to an oracle $\mathcal{O}$ that can answer queries of the form "Does (u, v) belong to M?" for every edge (u, v) of the graph. To find out if a node v is matched in M, it suffices to query the oracle if any of the edges incident to v is in M. It follows from the Hoeffding bound that $|M|$ can be estimated with constant probability and additive error at most εn by checking for $O(1/\varepsilon^2)$ randomly chosen vertices v, if they are matched in M.

Our main contribution is a new way of implementing oracles such the above for several problems. We now present our implementation of $\mathcal{O}$, for Maximal Matching. A random number $r_e \in [0, 1]$ is assigned to each edge e of the graph[1]. In order to decide if an edge q is in the matching, the oracle first determines the set of edges

[1] In an implementation, we do not assign all numbers r_e at the beginning. We can postpone assignment of the random number r_e to an edge e until we see it. After learning r_e, we keep it in case we need it later again. Moreover, arbitrary random real numbers in $[0, 1]$ cannot be generated in practice. Nevertheless, it suffices to discretize

adjacent to q of numbers r_e smaller than that of q, and recursively checks if any of them is in the matching. If at least one of the adjacent edges is in the matching, q is not in the matching; otherwise, it is.

Why does this work? Consider first the following trivial greedy algorithm for finding a maximal matching. The algorithm starts with an empty matching M. For each edge e, it checks if there is already an adjacent edge in M. If there is no such edge, it adds e to M. The final M is clearly a maximal matching, since every edge not in M is adjacent to at least one of the edges in M. Our oracle simulates this algorithm, considering edges in random order (which is generated by the random numbers r_e).

Query Complexity. It remains to bound the query complexity of the algorithm. Consider a chain of recursive calls. Note that in each consecutive recursive call the random number assigned to an edge decreases. Intuitively, on average it is divided by two. When it gets below, say, $1/(10d)$, then, since each edge is adjacent to at most $2d-2$ other edges, the probability that any edge must be considered becomes smaller than $1/5$. In this case, few random calls will be done with high probability.

Note that in our case, each query to the oracle is independent of the replies to the previous queries. Given this property, we prove in Appendix A that the expected number of queries to the graph that the oracle $\mathcal{O}$ makes in order to handle a single query is at most $2^{O(d)}$. This implies that the expected query complexity of the whole algorithm is at most $O(d/\varepsilon^2) \cdot 2^{O(d)} = 2^{O(d)}/\varepsilon^2$.

Lemma 1 *There is an algorithm of query complexity $2^{O(d)}/\varepsilon^2$ that given query access to a graph on n vertices of degree bounded by d, with probability at least $2/3$ computes $\hat{t}$ such that $t \leq \hat{t} \leq t + \varepsilon n$, where t is the size of some maximal matching in the graph.*

Immediate Corollary. Gavril (see [6]) proved that the set of nodes matched in any maximal matching is a proper vertex cover of size at most 2 times the optimum. Furthermore, it is also well known that the size of a maximal matching is at least one half of the maximum matching size. We immediately obtain approximation algorithms for two problems.

Corollary 2 *There are approximation algorithms of query complexity $2^{O(d)}/\varepsilon^2$ for the minimum vertex cover size and the maximum matching size for graphs of maximum degree at most d that with probability $2/3$ return $\hat{t}$ such that $t \leq \hat{t} \leq 2t+\varepsilon n$, where t is the minimum vertex cover size or the maximum matching size, respectively.*

the range $[0, 1]$ so that two edges are assigned the same number with negligibly small probability.

1.2. Technique High-Level Description

We now give a high-level overview of our technique, which we applied in a specific setting in Section 1.1. We also briefly describe conditions that must be met in order to make our technique applicable.

Transformation. Our technique transforms an algorithm $\mathcal{A}$ that computes an approximate solution to a problem into a constant-time algorithm that approximates the size of an optimal solution, provided $\mathcal{A}$ meets certain criteria. We require that $\mathcal{A}$ compute the approximate solution in a constant number of phases such that each phase is an application of any maximal set of disjoint local improvements. (The local improvements considered in each phase may be different.) Moreover, to achieve a constant running time, we require that each local improvement considered in a given phase intersect with at most a constant number of other considered local improvements. For example, the maximal matching algorithm of Section 1.1 constructs a maximal matching in just one phase, by taking a maximal set of non-adjacent edges.

The general idea behind the new constant-time algorithm that we construct is the following. Let k be the number of phases in the algorithm. For the i-th phase, where $1 \leq i \leq k$, we construct an oracle $\mathcal{O}_i$ that implements query access to the intermediate solution constructed by the i-th phase of the algorithm. ($\mathcal{O}_0$ gives the initial solution that the algorithm starts with.) $\mathcal{O}_i$ is itself given query access to $\mathcal{O}_{i-1}$, and simulates the i-th phase of the algorithm on the output of the $(i-1)$-st phase. Finally, $\mathcal{O}_k$ provides query access to a solution that $\mathcal{A}$ could output. By using random queries to $\mathcal{O}_k$, we approximate the size of the solution computed by $\mathcal{A}$.

Locality Lemma. We show that for the problems that we consider, the answer to a query about the output of the i-th phase can be computed, using in most cases only a small number of queries about the output of the $(i-1)$-st phase. We show that long chains of dependencies between prospective improvements in the i-th phase can be avoided by considering them in random order. Hence, it usually suffices to query a small neighborhood of each query point. We state this key result, which we refer to as "the Locality Lemma", in Section 2.

The Locality Lemma applied to our sample problem shows that with probability at least $2/3$ the algorithm returns a correct estimate and the number of queries is bounded by $2^{O(d^4)}/\varepsilon^4$. This is worse than the more specialized upper bound $2^{O(d)}/\varepsilon^2$ in Appendix A. The improved upper bound takes advantage of the fact that the algorithm consists of only one phase, and avoids dependencies between queries to oracles on lower levels.

1.3. Approximation Notion

Before we quantitatively list our results, we define the notion of approximation that we use. This notion combines additive and multiplicative approximation, and appears in theoretical computer science in many contexts.

Definition 3 *We say that a value $\hat{y}$ is an (α, β)-*approximation *to y, if*

$$y \leq \hat{y} \leq \alpha \cdot y + \beta.$$

*We say that an algorithm A is an (α, β)-*approximation algorithm *for a value $V(x)$ if it computes an (α, β)-approximation to $V(x)$ with probability at least $2/3$ for any proper input x.*

In all problems considered by us, under very natural assumptions, this notion suffices to get a multiplicative approximation. For instance, we later show a $(1, \varepsilon n)$-approximation algorithm for the maximum matching size for graphs of maximum degree at most d. If there are $\Omega(n)$ edges in the graph, the maximum matching size is at least $\Omega(n/d)$. Hence, by picking $\varepsilon = \Theta(\varepsilon'/d)$, we can get a multiplicative $(1+\varepsilon')$-approximation to the maximum matching size.

1.4. Model

In the paper we only deal with two different kinds of inputs. When the input is a graph, we assume query access to the adjacency list of each node, i.e., for each vertex v, we can ask which vertex is the i-th neighbor of v. In the set cover problem, we are given a set of sets S_i. For each S_i, we have access to a list of elements of S_i, and for each element, we have access to a list of sets it is in.

1.5. Our Results and Previous Work

We now list our results, and compare them with previous work in related areas. We express the complexity of our algorithms in terms of the necessary number of queries, but the corresponding running time is independent of the size of input and at most polynomial in the query complexity.

Vertex Cover. We show that there exists a $(2, \varepsilon n)$-approximation algorithm of query complexity $2^{O(d)}/\varepsilon^2$, for graphs of maximum degree bounded by d. Combining the results of Parnas and Ron [14] and Marko and Ron [13] yields a $(2, \varepsilon n)$-approximation algorithm for the minimum vertex cover size of running time and query complexity $d^{O(\log(d/\varepsilon))}$. Our algorithm has better dependency on ε, but worse on d. Furthermore, Trevisan showed that for any constant $c \in [1, 2)$, a $(c, \varepsilon n)$-approximation algorithm must use at least $\Omega(\sqrt{n})$ queries (the result appeared in [14]).

Maximum Matching. The only results on approximation of the maximum matching size in sublinear time that have been known before are the algorithms of Parnas and Ron [14]. Since their algorithms give a constant factor approximation to the minimum vertex cover size, they also give a constant factor approximation to the maximum matching size. The main obstacle to applying the general reduction of Parnas and Ron from distributed algorithms is that the known distributed algorithms [4, 5] for maximum matching run in a number of rounds that is polylogarithmic in the graph size, not constant.

We show that nevertheless, there exists a $(1, \varepsilon n)$-approximation algorithm for graphs of maximum degree bounded by $d \geq 2$ with query complexity $2^{d^{O(1/\varepsilon)}}$.

Maximum Weight Matching. For bounded-degree weighted graphs of all weights in $[0, 1]$, one can also show an algorithm that computes a $(1, \varepsilon n)$-approximation to the maximum weight matching with a number of queries that only depends on d and ε. This can be achieved by locally simulating an algorithm of Pettie and Sanders [15]. We omit this result in this version of the paper.

Set Cover. Let $H(i)$ be the i-th harmonic number $\sum_{1 \leq j \leq i} 1/j$. Recall that $H(i) \leq 1 + \ln i$. We show that there is an $(H(s), \varepsilon n)$-approximation algorithm of query complexity $\left(\frac{2^{(st)^4}}{\varepsilon}\right)^{O(2^s)}$ for the minimum set cover size, for instances with n sets S_i, each of size at most s, and with each element in at most t different sets. As a special case of the set cover problem, we get an $(H(d+1), \varepsilon n)$-approximation algorithm of query complexity $\left(\frac{2^{d^8}}{\varepsilon}\right)^{O(2^d)}$ for the minimum dominating set size for graphs of degree bounded by d.

Combining the results of Parnas and Ron [14] and Kuhn, Moscibroda and Wattenhofer [11] yields an

$(O(\log d), \varepsilon n)$-approximation algorithm for the minimum dominating set of query complexity $d^{O(\log d)}/\varepsilon^2$.

Other Problems. Another example of a problem in the same model is approximation of the minimum spanning tree weight for graphs of degree bounded by d. Chazelle, Rubinfeld and Trevisan [2] showed that if all edge weights are in $\{1, 2, \ldots, w\}$, then there is an algorithm that computes a multiplicative $(1+\varepsilon)$-approximation to the minimum weight spanning tree in time $O(\frac{dw}{\varepsilon^2} \log \frac{w}{\varepsilon})$.

1.6. Other Aspects

Connection to Property Testing. We consider the class $\mathcal{C}_d$ of graphs of the maximum degree bounded by d. In property testing of graphs in the bounded degree model [7], one wants to distinguish two subsets of $\mathcal{C}_d$: graphs that have a property P and those that need to have at least εdn edges added or removed to have P, where $\varepsilon > 0$ is a parameter.

Consider the property of having a perfect matching (we focus on graphs with an even number of nodes). Clearly, for a graph with a perfect matching, the maximum matching size is $n/2$. On the other hand, for any graph that must have at least εdn edges added or removed to have P, the maximum matching size is smaller than $n/2 - \Omega(\varepsilon dn)$. Our maximum matching algorithm can then be used to efficiently solve the testing problem in constant time.

Separation between Vertex Cover and Maximum Matching. Our result exhibits a separation between approximation of the minimum vertex cover size and approximation of the maximum matching size. In the standard Turing machine computation model, the former is known to be NP-hard, and the latter is computable in polynomial time. Note that this separation is conditional, since it is based on the $\mathrm{P} \neq \mathrm{NP}$ assumption, and the two values are always within a factor of 2. Here, in a natural computation model, we are able to prove an unconditional separation. A result due to Trevisan (see [14]) shows that approximation of the minimum vertex cover size to within a factor less than 2 requires $\Omega(\sqrt{n})$ queries, and we show that the size of the maximum matching can be approximated arbitrarily well in time independent of n.

Bounded Average-Degree Instances. Parnas and Ron [14] observed that one can easily turn some algorithms for bounded maximum-degree graphs into algorithms for bounded average-degree graphs. Consider the maximum matching problem. If the average degree is at most $\hat{d}$, then there are at most εn vertices of degree greater than $\hat{d}/\varepsilon$. Removing them from the graph changes the maximum matching size by at most εn. Therefore, running the algorithm for bounded maximum degree and ignoring all vertices of degree greater than $\hat{d}/\varepsilon$ gives a $(1, 2\varepsilon)$-approximate solution.

Distributed Algorithms. Our technique can be used to construct distributed algorithms that with constant probability produce a good solution for graph problems that we have considered. For instance, for the maximum matching problem, there is an algorithm that in $c = c(\varepsilon, d)$ communication rounds collects information about all vertices and edges in the radius c, and random numbers assigned to each prospective augmenting path. For all but a constant fraction of edges, the knowledge suffices to decide if they are in the matching or not. If the radius-c neighborhood does not suffice to decide if an edge is in the matching, we decide it is not. With high probability, only a small fraction of edges that should be in the matching is not included.

2. Locality Lemma

We now state and prove the Locality Lemma, which is the main tool for bounding query complexity in this paper. Let us first give its informal explanation. We are given a graph of bounded degree with random numbers $r(v)$ in $[0, 1]$ assigned independently to each node v of the graph. A function f is defined inductively on the nodes of the graph. The value of f at a node v is a function of only v and values of f at neighbors w of v such that $r(w) < r(v)$. The value of f at a node can be computed recursively. Suppose that we have an algorithm that does not know the numbers $r(v)$, and only wants to learn f at q different nodes. The lemma gives a bound which holds with high probability on the total number of nodes for which we must compute f in all recursive calls. A single phase of each of our algorithms can be expressed as this type of computation for a certain graph.

Lemma 4 (Locality Lemma) *Let $G = (V, E)$ be a graph of degree bounded by $d \geq 2$, and let $g : V \times (V \times A)^\star \to A$ be a function. A random number $r(v) \in [0, 1]$ is independently and uniformly assigned to each vertex v of G. A function $f_r : V \to A$ is defined recursively, using g. For each vertex v, we have*

$$f_r(v) = g(v, \{(w, f_r(w)) : r(w) < r(v)\}).$$

Let $\mathcal{A}$ be an algorithm that initially does not know r, but can adaptively query f_r at q different nodes. To

answer $\mathcal{A}$'s queries, it suffices to recursively compute $f_r(v)$ for at most

$$\frac{q^2}{\delta} \cdot C^{d^4}$$

nodes v with probability at least $1-\delta$, for any $\delta > 0$, where C is an absolute constant.

Proof Handling each query of $\mathcal{A}$ requires computing $f_r(v)$ for certain nodes v. Unfortunately, since $\mathcal{A}$'s queries may be adaptive, $\mathcal{A}$ may be able to deduce from answers to previous queries how to query f_r to increase the number of nodes for which the function must be recursively computed. Intuitively, if all $\mathcal{A}$'s query points were far away from each other, the sets of nodes explored to answer each query would likely be disjoint, and we could bound the number of nodes explored for each of them independently. Therefore, whenever we bound the number of $f_r(v)$ computed for $\mathcal{A}$'s query, we also make sure that there is no node w close to the query point such that computing $f_r(w)$ requires computing many $f_r(v)$.

We now give a few auxiliary definitions. We say that a node v can be *reached* or is *reachable* from a node w if there is a path $u_0 = w, u_1, \ldots, u_k = v$, such that $r(u_{i-1}) > r(u_i)$, for all $1 \leq i \leq k$. Less formally, v can be reached from w if we need to compute $f_r(v)$ in order to compute $f_r(w)$. The *reachability radius* of node v is the maximum distance between v and a node that is reachable from v.

What is the probability that for a given node v, the reachability radius is greater than t? The probability can be bounded by the probability that there is a path of length $t+1$ that starts at v, and the values r are strictly decreasing along the path. There are at most $d \cdot (d-1)^t$ such paths, and by symmetry, the probability that the values r decrease along a fixed such path is $1/(t+2)!$ Hence the probability of the event is at most $\frac{d(d-1)^t}{(t+2)!}$.

We now consider not just a single query, but all the (potentially adaptive) queries constructed by $\mathcal{A}$. What is the probability that for each query $f_r(w)$, the reachability radius of w is at most t? After each query, $\mathcal{A}$ learns something about r, and may use this knowledge to create a malicious query. Note that it cannot learn more than $r(v)$ for nodes v that are either reachable from w or are a neighbor of a node reachable from w, where one of the queries was about $f_r(w)$. Suppose that the reachability radius for all previous query points was at most t. Let v be a vertex at distance greater than $2(t+1)$ from each of the previous query points. The probability that the reachability radius of v is at most t depends only on $r(u)$, for u at distance at most $t+1$ from v, and hence is independent of the algorithm's current knowledge. We only need to make sure that for a query about $f(v)$, for v at distance at most $2(t+1)$ to one of the previous query points, v has small reachability radius. This may depend on the knowledge of the algorithm. Hence, for any query point v, we also bound the reachability radius for all vertices at distance at most $2(t+1)$, so that the algorithm is unlikely to construct a query that requires exploring many vertices.

For each query point v, there are at most

$$1 + d\sum_{i=0}^{2t+1}(d-1)^i \leq 1 + d\sum_{i=0}^{2t+1} d^i \leq d^{2t+3}$$

vertices at distance at most $2(t+1)$. The total number of nodes close to one of the q query points is hence at most $q \cdot d^{2t+3}$. We want to bound the reachability radius by t for all of them. By the union bound, the probability that the algorithm queries some $f_r(v)$, where v has reachability radius greater than t is at most

$$qd^{2t+3} \cdot \frac{d(d-1)^t}{(t+2)!} \leq \frac{q \cdot d^{3t+4}}{(t+2)!} \leq q \cdot \left(\frac{3d^3}{t+2}\right)^{t+2}.$$

Let E_i be the event that the maximum reachability radius for all query points is exactly i, and $E_{>i}$ the event that it is greater than i. We have, $\Pr[E_{>i}] \leq q \cdot \left(\frac{3d^3}{i+2}\right)^{i+2}$.

What is the expected total number T of vertices for which we recursively compute f_r? It is

$$\begin{aligned} T &\leq \sum_{i\geq 0} \Pr[E_i] \cdot q(1 + d \cdot \sum_{0\leq j\leq i-1}(d-1)^j) \\ &\leq \sum_{i\geq 0} \Pr[E_i] \cdot qd^{i+1} \leq \sum_{i\geq 0} \Pr[E_{>i-1}] \cdot qd^{i+1} \\ &\leq \sum_{i\geq 0} q\left(\frac{3d^3}{i+1}\right)^{i+1} \cdot qd^{i+1} \leq q^2 \sum_{i\geq 0}\left(\frac{3d^4}{i+1}\right)^{i+1}. \end{aligned}$$

For $i \geq 6d^4 - 1$, we have

$$\sum_{i\geq 6d^4-1}\left(\frac{3d^4}{i+1}\right)^{i+1} \leq \sum_{i\geq 6d^4-1} 2^{-(i+1)} \leq 1.$$

Using calculus, one can show that the term $\left(\frac{3d^4}{i+1}\right)^{i+1}$ is maximized for $i+1 = \frac{3d^4}{e}$, and hence,

$$\sum_{i<6d^4-1}\left(\frac{3d^4}{i+1}\right)^{i+1} \leq (6d^4-1) \cdot e^{\frac{3d^4}{e}}.$$

We get

$$\begin{aligned} T &\leq q^2\left(1+(6d^4-1)\cdot e^{\frac{3d^4}{e}}\right) \\ &\leq q^2\cdot 6d^4\cdot e^{\frac{3d^4}{e}} \leq q^2C^{d^4}, \end{aligned}$$

for some constant C. By Markov's inequality, the probability that the number of queries is greater than T/δ is at most δ. Hence with probability at least $1-\delta$, the number of queries is bounded by $q^2C^{d^4}/\delta$. ■

3. Maximum Matching

3.1. Definitions and Notation

Let M be a *matching* in a graph $G=(V,E)$, that is, a subset of nonadjacent edges of G. A node v is *M-free* if v is not an endpoint of an edge in M. A path P is an *M-alternating* path if it consists of edges drawn alternately from M and from $E\setminus M$. A path P is an *M-augmenting* path if P is M-alternating and both endpoints of P are M-free nodes (i.e., $|P\cap M| = |P\cap(E\setminus M)|+1$).

3.2. Properties of Matchings

Let $\oplus$ denote the symmetric difference of sets. If M is a matching and P is an M-augmenting path, then $M\oplus P$ is a matching such that $|M\oplus P| = |M|+1$. Many matching algorithms search for augmenting paths until they construct a maximum matching, and one can show that in an non-maximum matching there is an augmenting path.

The correctness of our algorithm relies on the properties of matchings proven by Hopcroft and Karp [8]. The part of their contribution that is important to us is summarized below.

Fact 5 (Hopcroft and Karp [8]) *Let M be a matching with no augmenting paths of length smaller than t. Let $P^\star$ be a maximal set of vertex-disjoint M-augmenting paths of length t. Let A be the set of all edges in the paths in $P^\star$. There does not exist an $(M\oplus A)$-augmenting path of length smaller than or equal to t.*

We now prove an auxiliary lemma that connects the minimum length of an augmenting path and the quality of the matching.

Lemma 6 *Let M be a matching that has no augmenting paths of length smaller than $2t+1$. Let $M^\star$ be a maximum matching in the same graph. It holds $|M| \geq \frac{t}{t+1}|M^\star|$.*

Proof Consider the set of edges $\Delta = M\oplus M^\star$. There are exactly $|M^\star|-|M|$ more edges from $M^\star$ then from M in Δ. Since M and $M^\star$ are matchings, each vertex is incident to at most two edges in Δ. Hence Δ can be decomposed into paths and cycles. Each path of even length and each cycle contain the same number of edges from M and $M^\star$. Each path P of odd length contains one more edge from $M^\star$ than from M. It if contained one more edge from M, it would be an $M^\star$-augmenting path; an impossibility. P is then an M-augmenting path. Summarizing, we have exactly $|M^\star|-|M|$ odd-length vertex-disjoint paths in Δ, and each of them is an M-augmenting path.

Since each M-augmenting path has length at least $2t-1$, this implies that $|M| \geq t(|M^\star|-|M|)$. Hence, $|M| \geq \frac{t}{t+1}|M^\star|$. ■

3.3. The Algorithm

Consider the maximum matching problem in an unweighted graph of bounded degree d. It is well known that the size of any maximal matching is at least half of the maximum matching size. Because of that, we obtained a $(2,\varepsilon n)$-approximation algorithm for the maximum matching size in Corollary 2. We now show that our technique can be used to achieve better approximations in constant time.

A Sequential Algorithm. We simulate the following sequential algorithm. The algorithm starts with an empty matching M_0. In the i-th phase, it constructs a matching M_i from M_{i-1} as follows. Let $P^\star_{i-1}$ be a maximal set of vertex-disjoint M_{i-1}-augmenting paths of length $2i-1$. Let A_{i-1} be the set of all edges in the augmenting paths in $P^\star_{i-1}$. We set $M_i = M_{i-1}\oplus A_{i-1}$. If M_{i-1} is a matching, so is M_i. By induction, all M_i are matchings. The algorithm stops for some k, and returns M_k.

We now show that M_i has no augmenting path of length smaller than $2i+1$. M_1 is a maximal matching, so it has no augmenting path of length smaller than 3. Now, for the inductive step, assume that M_{i-1}, $i\geq 1$, has no augmenting path shorter than $2i-1$. $P^\star_{i-1}$ is a maximal set of vertex-disjoint M_{i-1}-augmenting paths of length $2i-1$. Therefore, it follows by Fact 5 that M_i does not have any augmenting path shorter than $2i+1$.

Set $k=\lceil 1/\delta\rceil$, and let $M^\star$ be a maximum matching. By Lemma 6, $\frac{k}{k+1}|M^\star| \leq |M_k| \leq |M^\star|$, which yields $|M^\star| \leq \frac{k+1}{k}|M_k| \leq (1+\delta)|M^\star|$. If we had an estimate α such that $2|M_k| \leq \alpha \leq 2|M_k|+\varepsilon n/2$, we could get a $(1+\delta,\varepsilon n)$-approximation to $|M^\star|$ by multiplying α by $\frac{k+1}{2k}$, which is at most 1.

The Constant-Time Algorithm. We construct a sequence of oracles $\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_k$. A query to $\mathcal{O}_i$ is an edge $e \in E$. The oracle's reply indicates whether e is in M_i. To compute the required α, it suffices to estimate the fraction of vertices that are matched in $|M_k|$. In order to do so, one can sample $O(1/\varepsilon^2)$ vertices, and for each of them, check if any incident edge is in M_k or not. The correctness of the estimate with probability $5/6$ follows from the Hoeffding bound.

The oracles $\mathcal{O}_i$ are constructed by using our technique for transforming algorithms into constant-time algorithms. $\mathcal{O}_i$ has access to $\mathcal{O}_{i-1}$, and simulates the i-th phase of the above algorithm. We assume that each M_{i-1}-augmenting path P of length $2i-1$ is assigned a random number $r(P)$, which is uniformly and independently chosen from $[0,1]$. These random numbers give a random ordering of all the M_{i-1}-augmenting paths. $P^\star_{i-1}$ is the greedily constructed maximal set of vertex-disjoint M_{i-1}-augmenting paths P considered in order of their $r(P)$. To handle a query about an edge e, the oracle first finds out if $e \in M_{i-1}$, and then, checks if there is an M_{i-1}-augmenting path in $P^\star_{i-1}$ that contains e. If there is such a path, the answer of $\mathcal{O}_i$ to the query about e is the opposite of the answer of $\mathcal{O}_{i-1}$. Otherwise, it remains the same.

The oracle can easily learn all length-$(2i-1)$ M_{i-1}-augmenting paths that contain e by querying G and O_{i-1}. To find out which augmenting paths are in $P^\star_{i-1}$, the oracle considers the following graph H_i. All the M_{i-1}-augmenting paths of length $2i-1$ are nodes of H_i. Two nodes P_1 and P_2 are connected in H_i if P_1 and P_2 share a vertex. To check if P is in $P^\star_{i-1}$, it suffices to check if any of the paths R corresponding to the vertices adjacent to P in H_i is in $P^\star_{i-1}$, for $r(R) < r(P)$. If none, $P \in P^\star_{i-1}$. Otherwise, P is not in $P^\star_{i-1}$. This procedure can be run recursively. This finishes the description of the algorithm.

Query Complexity. It remains to bound the number of queries of the entire algorithm to the graph. This is accomplished in the following lemma.

Lemma 7 *The number of queries of the algorithm is with probability $5/6$ of order $\frac{2^{O(d^{9k})}}{\varepsilon^{2^{k+1}}}$, where $k = \lceil 1/\delta \rceil$, and $d \geq 2$ is a bound on the maximum degree of the input graph.*

Proof Our main algorithm queries $\mathcal{O}_k$ about edges adjacent to C'/ε^2 random vertices, where C' is a constant. Let $Q_{k+1} = C' \cdot d/\varepsilon^2$ be the number of the direct queries of the main algorithm to G. These queries are necessary to learn the edges that $\mathcal{O}_k$ is queried with. Let Q_{i+1} be an upper bound on the number of queries of the algorithm to $\mathcal{O}_i$. We now show an upper bound Q_i on the number of queries to G performed by $\mathcal{O}_i$. The upper bound holds with probability at least $1 - \frac{1}{6k}$. Q_i also bounds the number of queries to O_{i-1}, since $\mathcal{O}_i$ does not query any edge it has not learnt about from G. For each received query about an edge e, $\mathcal{O}_i$ first learns all edges within the distance of $2i-1$ from e, and checks which of them are in M_{i-1}. For a single e, this can be done with at most $d \cdot 2\sum_{j=0}^{2i-2}(d-1)^j \leq 2d^{2i}$ queries to both G and $\mathcal{O}_{i-1}$, and suffices to find all length-$(2i-1)$ M_{i-1}-augmenting paths that contain e.

There are at most id^{2i-1} length-$(2i-1)$ paths in G that contain a fixed vertex v. Each such path can be broken into two paths that start at v. The length of the shorter is between 0 and $i-1$, and there are at most d^t paths of length t that start at t.

The number of length-$(2i-1)$ M_{i-1}-augmenting paths that contain e is therefore at most id^{2i-1}. Moreover, the maximum degree of H_i can be bounded by the number of length-$(2i-1)$ paths that intersect a given length-$(2i-1)$ augmenting path. Hence, the degree of H_i is at most $2i \cdot id^{2i-1} = 2i^2 d^{2i-1}$. Finally, to find augmenting paths adjacent in H_i to a given augmenting path P, it suffices to learn whether e' is in M_{i-1}, for each edge e' within the radius of $2i$ from any of the vertices of P. This can be accomplished with at most $2i \cdot d\sum_{j=0}^{2i-1} d^j \leq 2id^{2i+1}$ queries to both G and $\mathcal{O}_{i-1}$.

In total, to answer queries about all, at most Q_{i+1} edges e, $\mathcal{O}_i$ must check membership in $P^\star_{i-1}$ for at most $Q_{i+1} \cdot 2id^{2i-1}$ augmenting paths. By the Locality Lemma, the number of augmenting paths for which we recursively check membership in $P^\star_{i-1}$ is with probability $1 - \frac{1}{6k}$ at most

$$(Q_{i+1} \cdot id^{2i-1})^2 \cdot C^{\left(2i^2 d^{2i-1}\right)^4} \cdot 6k \leq 2^{O\left(d^{8i}\right)} \cdot kQ_{i+1}^2.$$

For each of them we compute all adjacent paths in H_i. Therefore, with probability $1 - \frac{1}{6k}$, the total number of $\mathcal{O}_i$'s queries to both $\mathcal{O}_{i-1}$ and G is bounded by

$$Q_{i+1} \cdot 2d^{2i} + 2^{O\left(d^{8i}\right)} \cdot kQ_{i+1}^2 \cdot 2id^{2i+1}$$

$$\leq 2^{O\left(d^{8i}\right)} \cdot kQ_{i+1}^2 =: Q_i.$$

The total number of queries to G in the entire algorithm can be bounded by

$$\sum_{j=1}^{k+1} Q_j \leq 2Q_1 \leq \left(\frac{C' \cdot d \cdot k}{\varepsilon^2}\right)^{2^k} \cdot 2^{O\left(d^{8k}\right) \cdot 2^k}$$

$$\leq \frac{2^{O(d^{9k})}}{\varepsilon^{2^{k+1}}}. \qquad \blacksquare$$

We summarize the whole algorithm in the following theorem.

Theorem 8 *There is a $(1+\delta, \varepsilon n)$-approximation algorithm for the maximum matching size that uses $\frac{2^{O(d^{9k})}}{\varepsilon^{2k+1}}$ queries, where $d \geq 2$ is a bound on the maximum degree, and $k = \lceil 1/\delta \rceil$.*

Proof We run the algorithm described above. If the algorithm exceeds the number of queries guaranteed in Lemma 7, we terminate it, and return an arbitrary result. The algorithm returns a $(1+\delta, \varepsilon n)$-approximation with probability at least $2/3$, because the sampling can return a wrong estimate with probability at most $1/6$, and the algorithm can exceed the allowed number of queries with probability at most $1/6$. ■

Finally, we can easily remove the multiplicative factor.

Corollary 9 *There is a $(1, \varepsilon n)$-approximation algorithm of query complexity $2^{d^{O(1/\varepsilon)}}$ for the maximum matching size, where $d \geq 2$ is a bound on the maximum degree.*

Proof Using the algorithm of Theorem 8, we can get a $(1+\varepsilon, \varepsilon n/2)$-approximation for the maximum matching size, using $2^{d^{O(1/\varepsilon)}}$ queries. Since the size of the maximum matching is at most $n/2$, this approximation is also a $(1, \varepsilon n)$-approximation for the maximum matching size. ■

4. Set Cover

In the minimum set cover problem, an input consists of subsets S_1 to S_n of $U = \{1, \ldots, m\}$. Each element of U belongs to at least one of the sets S_i. The goal is to cover U with the minimum number of sets S_i, that is, to find a minimum size set I of indices such that $\bigcup_{i \in I} S_i = U$. In this paper, we want to approximate the optimal size of I.

We assume that for each set S_i, we have query access to a list of elements of S_i, and that for each element $u \in U$, we have query access to a list of indices of sets S_i that contain u.

4.1. The Classical Greedy Algorithm

Theorem 10 *There is an $(H(s), \varepsilon n)$-approximation algorithm of query complexity $\left(\frac{2^{(st)^4}}{\varepsilon}\right)^{O(2^s)}$ for the minimum set cover size for instances with all n sets S_i of size at most s, and each element in at most t different sets.*

Proof We simulate the classical greedy algorithm [10, 12] for the set cover problem. The algorithm starts from an empty cover, and keeps adding the set S_i which covers most elements that have not yet been covered, until the whole set U is covered. The approximation factor of the algorithm is at most $H(s) \leq 1 + \ln s$.

We first consider all sets in random order and add to the cover those that cover s new elements at the time they are considered. Let $\mathcal{C}_1$ be the set of sets that were already included into the cover. We then consider the remaining sets, also in random order, and we add to the cover those that cover $s-1$ new elements. This way we get $\mathcal{C}_2$, the set of all sets already included in the cover. We keep doing this until we cover the whole set U, and $\mathcal{C}_s$ is the final cover. We show that in most cases one can check if a set is in the cover without simulating the whole process.

We create a sequence of oracles $\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_s$ that correspond to the process described above. A query to an oracle $\mathcal{O}_j$ is an index i of a set S_i. The oracle's reply indicates whether S_i is in $\mathcal{C}_j$.

How is $\mathcal{O}_j$ implemented? We assume that each set S_i is assigned a random number r_{ji}, which is uniformly and independently chosen from $[0, 1]$. These random numbers give a random ordering of sets S_i. To handle a query about a set S_k, we first ask $\mathcal{O}_{j-1}$ if S_k was already included in $\mathcal{C}_{j-1}$. (For $j = 1$, we assume that $\mathcal{C}_{j-1} = \mathcal{C}_0 = \emptyset$, so no query is necessary in this case.) If S_k was already in $\mathcal{C}_{j-1}$, then it is also in $\mathcal{C}_j$. Otherwise, we learn first the elements of S_k (at most s queries) and what sets S_i they belong to (at most st further queries). Then, we check for each of these S_i if it was already in $\mathcal{C}_{j-1}$ (at most st queries to $\mathcal{O}_{j-1}$), and for all of the S_i's that have $r_{ji} < r_{jk}$, we recursively check if they are in $\mathcal{C}_j$. Finally, using this knowledge, we can verify what number of elements of S_k is not yet covered, when S_k is considered. If the number of these elements is $s - j + 1$, then S_k is in $\mathcal{C}_j$. Otherwise, the number of the elements is lower than $s - j + 1$, and S_k is not in $\mathcal{C}_j$.

It is obvious that the above procedure simulates the classical greedy algorithm. Our sublinear approximation algorithm queries $\mathcal{O}_s$ for C'/ε^2 sets chosen at random, where C' is a sufficiently large constant, to estimate the fraction of sets which are in the cover to within $\varepsilon n/2$ with probability $5/6$. By adding $\varepsilon n/2$ to the estimate, we get the desired approximation. We want to bound the number of queries. We set Q_s to $(C'/\varepsilon^2)^2 \cdot 6s \cdot C^{(st)^4}$, and define Q_j, for $1 \leq j \leq s-1$, to be $Q_j \leq (Q_{j+1} \cdot (st+1))^2 \cdot 6s \cdot C^{(st)^4}$. By Lemma 4, each Q_i bounds the number of sets for which we check if they are in $\mathcal{C}_i$ with probability $1 - s \cdot \frac{1}{6s} = 1 - \frac{1}{6} = \frac{5}{6}$. It can be shown

by induction that

$$Q_{s-i} = \left(\frac{6s \cdot (st+1) \cdot C^{(st)^4}}{\varepsilon}\right)^{O(2^i)} = \left(\frac{2^{(st)^4}}{\varepsilon}\right)^{O(2^i)}$$

with probability at least $5/6$. So with probability $5/6$, the total number of queries is at most

$$(s+st) \cdot \sum_{i=1}^{s} Q_s = \left(\frac{2^{(st)^4}}{\varepsilon}\right)^{O(2^s)}.$$

Summarizing, with probability $2/3$, the algorithm uses the above number of queries, and returns the desired approximation. ■

4.2. Application to Dominating Set

In the dominating set problem, one is given a graph, and chooses a minimum size subset S of its vertices such that each vertex v of the graph is either in S or is adjacent to a vertex in S.

Theorem 11 *There is an $(H(d+1), \varepsilon n)$-approximation algorithm of query complexity $\left(\frac{2^{d^8}}{\varepsilon}\right)^{O(2^d)}$, for the minimum dominating set size for graphs with the maximum degree bounded by d.*

Proof The problem can be trivially expressed as an instance of the set cover problem. For each vertex v, we have a set S_v of size at most $d+1$ that consists of v and all neighbors of v. We want to cover the set of vertices of the graph with the minimum number of sets S_v. To approximate the minimum set cover size, we use the algorithm of Theorem 10. ■

4.3. Query Lower Bound Discussion

Trevisan proved that for every two positive constants γ and ϵ, there exists a constant d such that obtaining with constant probability a $(2-\gamma, \varepsilon n)$-approximation of the size of a minimum vertex cover of graphs over n vertices and degree d, requires $\Omega(\sqrt{n})$ queries [14]. Note that vertex cover is a special case of set cover, in which we want to cover the set of edges of the graph. For each vertex v, we create a set S_v that consists of edges incident to v. This implies that the above lower bound also applies to set cover, and shows that it is impossible to get a constant approximation factor less than 2 with the number of queries independent of the input size.

5. Future Research

It is interesting to understand what other approximation problems our technique applies to. We also believe that the locality lemma can be improved.

Moreover, it is likely that our analysis of the maximal matching algorithm is suboptimal. In particular, if we consider edges adjacent to a given edge in order of their random numbers, then after we learn that a single edge is in the maximal matching, we do not have to consider edges adjacent to this edge, as they cannot be in the matching. Our experiments indicate that the number of queries may grow quasi-polynomially, not exponentially, in d. If this is in fact the case, our algorithm may actually outperform the algorithm that follows from the results of Marko and Ron, and Parnas and Ron [13, 14] in all parameters.

Acknowledgement

The authors thank Ronitt Rubinfeld for her invaluable help in improving the presentation of the paper.

References

[1] M. Badoiu, A. Czumaj, P. Indyk, and C. Sohler. Facility location in sublinear time. In *ICALP*, pages 866–877, 2005.

[2] B. Chazelle, R. Rubinfeld, and L. Trevisan. Approximating the minimum spanning tree weight in sublinear time. In *ICALP*, pages 190–200, 2001.

[3] A. Czumaj and C. Sohler. Estimating the weight of metric minimum spanning trees in sublinear-time. In *STOC*, pages 175–183, 2004.

[4] A. Czygrinow and M. Hańćkowiak. Distributed algorithm for better approximation of the maximum matching. In *COCOON*, pages 242–251, 2003.

[5] A. Czygrinow, M. Hańćkowiak, and E. Szymańska. A fast distributed algorithm for approximating the maximum matching. In *ESA*, pages 252–263, 2004.

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

[7] O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.

[8] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[9] P. Indyk. Sublinear time algorithms for metric space problems. In *STOC*, pages 428–434, 1999.

[10] D. S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. Syst. Sci.*, 9(3):256–278, 1974.

[11] F. Kuhn, T. Moscibroda, and R. Wattenhofer. The price of being near-sighted. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 980–989, New York, NY, USA, 2006. ACM.

[12] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.

[13] S. Marko and D. Ron. Distance approximation in bounded-degree and general sparse graphs. In *APPROX-RANDOM*, pages 475–486, 2006.

[14] M. Parnas and D. Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, 381(1-3):183–196, 2007.

[15] S. Pettie and P. Sanders. A simpler linear time 2/3-epsilon approximation for maximum weight matching. *Inf. Process. Lett.*, 91(6):271–276, 2004.

A. Improved Query Complexity Analysis for Maximal Matching Size

Lemma 12 *Let G be a graph such that each vertex is adjacent to at most d other vertices. For each vertex v, we independently select a random number $r(v)$ from the range $[0,1]$. The expected query complexity of an algorithm that starts from a vertex u chosen independently of the values $r(v)$, and explores all paths $w_0 = u, w_1, \ldots, w_k$ such that $r(w_0) > r(w_1) > \ldots > r(w_k)$ is $2^{O(d)}$.*

Proof Let $Q(x)$ be the maximum expected number of queries for any vertex v, if $r(v) = x$. Obviously, if $x \leq y$, then $Q(x) \leq Q(y)$. Let $q_1, q_2, \ldots, q_k$ be neighbors of v, where $k \leq d$. For each i, $y_i = r(q_i)$ is a random variable. We claim that

$$Q(x) \quad \leq \quad d + \sum_{i=1}^{k} \Pr[y_i < x] \cdot E\left[Q(y_i) \,|y_i < x\right].$$

We first read the entire neighborhood of v with at most d queries. Then, for each neighbor q_i with $y_i < r$, we explore paths that start from q_i with the expected number of queries bounded by $Q(y_i)$.

In particular, for any $i \in \{1, \ldots, 2d\}$, we have

$$Q\left(\frac{i}{2d}\right) \leq d + \sum_{j=1}^{d} E\left[Q(y_j) \,\middle|\, y_j < \frac{i}{2d}\right] \cdot \Pr\left[y_j < \frac{i}{2d}\right]$$

$$\leq d + \sum_{j=1}^{d} \sum_{k=1}^{i} E\left[Q(y_j) \,\middle|\, y_j \in \left[\frac{k-1}{2d}, \frac{k}{2d}\right)\right] \cdot \Pr\left[y_j \in \left[\frac{k-1}{2d}, \frac{k}{2d}\right)\right]$$

$$\leq d + \sum_{j=1}^{d} \sum_{k=1}^{i} Q\left(\frac{k}{2d}\right) \cdot \frac{1}{2d} \leq d + \frac{1}{2} \sum_{k=1}^{i} Q\left(\frac{k}{2d}\right).$$

Hence,

$$Q\left(\frac{i}{2d}\right) \leq 2d + \sum_{k=1}^{i-1} Q\left(\frac{k}{2d}\right).$$

It can be shown by induction that

$$Q\left(\frac{i}{2d}\right) \leq 2d\left(2^i - 1\right).$$

It holds

$$Q\left(\frac{1}{2d}\right) \leq 2d \leq 2d\left(2^1 - 1\right),$$

and for $i \geq 2$,

$$\begin{aligned} Q\left(\frac{i}{2d}\right) \quad &\leq \quad 2d + \sum_{k=1}^{i-1} Q\left(\frac{k}{2d}\right) \\ &\leq \quad 2d\left(1 + \sum_{k=1}^{i-1} 2^k - 1\right) \\ &\leq \quad 2d\left(2^i - i + 1\right) \leq 2d\left(2^i - 1\right). \end{aligned}$$

Finally, we get

$$Q(r(u)) \leq Q(1) \leq 2d\left(2^{2d} - 1\right) = O\left(2^{O(d)}\right). \quad \blacksquare$$

Corollary 13 *The query complexity of the algorithm presented in Section 1.1 is $2^{O(d)}/\varepsilon^2$.*

Proof We generate in advance $O(1/\varepsilon^2)$ nodes to satisfy the Hoeffding bound. For all at most $O(d/\varepsilon^2)$ edges incident to them, we query oracle $\mathcal{O}$ if they are in the maximal matching or not. By linearity of expectation, the total expected number of queries equals the expectation of one of them times their number. Look at the dual graph G' of G. Each vertex of G' corresponds to an edge in G, and two vertices in G' are connected if and only if the corresponding edges in G are adjacent. Note that the maximum vertex degree in G' is at most $2d-2$. By Lemma 12, the oracle needs on average at most $2^{O(d)}$ queries to G' to compute an answer to a single query about a membership of an edge e in M. A single query to G' can easily be simulated with $O(d)$ queries to G. Hence, the total expected query complexity of the algorithm is

$$O\left(\frac{d}{\varepsilon^2}\right) \cdot 2^{O(d)} \cdot O(d) = \frac{2^{O(d)}}{\varepsilon^2}. \quad \blacksquare$$