SIAM J. COMPUT.
Vol. 38, No. 6, pp. 2526–2547

RANDOMIZED DIVIDE-AND-CONQUER: IMPROVED PATH, MATCHING, AND PACKING ALGORITHMS*

JIANER CHEN†, JOACHIM KNEIS‡, SONGJIAN LU†, DANIEL MÖLLE‡, STEFAN RICHTER‡, PETER ROSSMANITH‡, SING-HOI SZE§, AND FENGHUI ZHANG†

Abstract. We propose a randomized divide-and-conquer technique that leads to improved randomized and deterministic algorithms for NP-hard PATH, MATCHING, and PACKING problems. For the parameterized MAX-PATH problem, our randomized algorithm runs in time $O(4^k k^{2.7} m)$ and *polynomial space* (where m is the number of edges in the input graph), improving the previous best randomized algorithm for the problem that runs in time $O(5.44^k km)$ and *exponential space*. Our randomized algorithms for the parameterized MAX r-D MATCHING and MAX r-SET PACKING problems run in time $4^{(r-1)k} n^{O(1)}$ and *polynomial space*, improving the previous best algorithms for the problems that run in time $10.88^{rk} n^{O(1)}$ and *exponential space*. Moreover, our randomized algorithms can be derandomized to result in significantly improved deterministic algorithms for the problems, and they can be extended to solve other matching and packing problems.

Key words. randomized algorithm, divide-and-conquer, PATH, MATCHING, SET PACKING

AMS subject classifications. 68Q25, 68R05, 68W20, 68W40

DOI. 10.1137/080716475

1. Introduction.

This paper studies new and improved algorithmic techniques for exact and parameterized algorithms for NP-hard PATH, MATCHING, and PACKING problems, a research direction that has recently drawn considerable attention [2, 4, 7, 10, 11, 13, 15, 16, 20, 21].

1.1. Preliminaries and problem formulations.

Let G be an undirected graph. A *path* ρ in G is a sequence of vertices $\langle v_1, \ldots, v_k \rangle$ in G such that for all $1 \leq i \leq k-1$, $[v_i, v_{i+1}]$ is an edge in G, where k is called the *length* of ρ. The path ρ is *simple* if no vertex is repeated in the sequence. A *k-path* in G is a simple path of length k in G. If the graph G is weighted (i.e., if each vertex in G is assigned a *weight* that is a real number), then the *weight* of the path ρ is equal to the sum of weights of the vertices in ρ.

DEFINITION 1.1 (the parameterized maximum path problem (P-MAX PATH)). *Given a weighted undirected graph G of n vertices and m edges and a parameter k, either construct a k-path in G whose weight is the maximum over all k-paths in G or report that no k-path exists in G.*

There is an *unweighted version* for the P-MAX PATH problem in which the input

*Received by the editors February 25, 2008; accepted for publication (in revised form) February 24, 2009; published electronically May 13, 2009. Part of the results in this paper was reported, independently, at *The 18th Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA 2007) and at *The 32nd International Workshop on Graph-Theoretic Concepts in Computer Science* (WG 2006).
http://www.siam.org/journals/sicomp/38-6/71647.html

†Department of Computer Science & Engineering, Texas A&M University, College Station, TX 77843 (chen@cs.tamu.edu, sjlu@cs.tamu.edu, fhzhang@cs.tamu.edu). The work of these authors was supported in part by the National Science Foundation under grants CCR-0311590 and CCF-0830455.

‡Department of Computer Science, RWTH Aachen University, 52056 Aachen, Germany (kneis@cs.rwth-aachen.de, moelle@cs.rwth-aachen.de, richter@cs.rwth-aachen.de, rossmani@cs.rwth-aachen.de). The work of these authors was supported by the Deutsche Forschungsgemeinschaft of Germany under grants RO 927/7 and RO 927/6.

§Departments of Computer Science & Engineering and Biochemistry & Biophysics, Texas A&M University, College Station, TX 77843 (shsze@cs.tamu.edu). This author's work was supported in part by the National Science Foundation under grant CCR-0311590.

graph is unweighted. The unweighted version can be trivially reduced to the general version by regarding an unweighted graph as a weighted graph in which all vertices are assigned weight 1. There is also a version of the problem on directed graphs, where we are looking for a directed k-path of the maximum weight in a weighted and directed graph. We will be focused on the problem on undirected graphs and, in certain places, extend our discussion to directed graphs.

A set M of points in the r-dimensional Euclidean space R^r is a *matching* if no two points in M agree in any coordinate. A *k-matching* is a matching consisting of exactly k points in R^r. If each point in R^r is assigned a weight, then the *weight* of a matching M is equal to the sum of weights of the points in M.

DEFINITION 1.2 (the parameterized maximum r-dimensional matching problem (P-MAX r-D MATCHING)). *Given a set S of n points in the r-dimensional Euclidean space R^r and a parameter k, where each point in S is assigned a weight, either construct a k-matching in S whose weight is the maximum over all k-matchings in S or report that no k-matching exists in S.*

We also have an *unweighted version* for the P-MAX r-D MATCHING problem, which assumes that all points in the input set S are assigned weight 1.

An *r-set* is a set containing exactly r elements. A collection $\mathcal{P}$ of r-sets is a *packing* if no two r-sets in $\mathcal{P}$ intersect. A *k-packing* is a packing consisting of exactly k r-sets. If each r-set is assigned a weight, then the *weight* of a packing $\mathcal{P}$ is the sum of weights of the r-sets in $\mathcal{P}$.

DEFINITION 1.3 (the parameterized maximum r-set packing problem (P-MAX r-SET PACKING)). *Given a collection $\mathcal{C}$ of n r-sets and a parameter k, where each r-set in $\mathcal{C}$ is assigned a weight, either construct a k-packing in $\mathcal{C}$ whose weight is the maximum over all k-packings in $\mathcal{C}$ or report that no k-packing exists in $\mathcal{C}$.*

The *unweighted version* of the P-MAX r-SET PACKING problem assumes that all r-sets in the input collection $\mathcal{C}$ are assigned weight 1.

It is easy to see that the P-MAX r-D MATCHING problem can be trivially reduced to the P-MAX r-SET PACKING problem. Therefore, any algorithm solving the latter can be directly used to solve the former.

1.2. Previous work. Most previous algorithms for the P-MAX PATH, P-MAX r-D MATCHING, and P-MAX r-SET PACKING problems were presented for the unweighted versions of the problems. Many of these algorithms, with minor modifications, also work for the general versions of the problems.

The P-MAX PATH problem is closely related to a number of well-known NP-hard problems, such as the longest path, Hamiltonian path, and traveling salesman problems. Earlier algorithms [3, 16] for the problem have running time bounded by $2^k k! n^{O(1)}$. Papadimitriou and Yannakakis [19] studied a restricted version of the problem and conjectured that it is solvable in polynomial time to determine if a graph contains a $(\log n)$-path. This conjecture was confirmed by Alon, Yuster, and Zwick [2], who presented for the P-MAX PATH problem randomized and deterministic algorithms of running time $2^{O(k)} n^{O(1)}$. Recently, the P-MAX PATH problem has found applications in bioinformatics for detecting signaling pathways in protein interaction networks [21] and for biological subnetwork matchings [11].

The P-MAX r-D MATCHING and P-MAX r-SET PACKING problems were first studied by Downey and Fellows [6], where deterministic algorithms of time $(rk)!(rk)^{3rk} n^{O(1)}$ were developed. The upper bounds on the complexity of these problems were subsequently improved to $(r-1)^k((r-1)k/e)^{k(r-2)} n^{O(1)}$ [4, 10] based on the *greedy localization* techniques [4]. Koutis [13] developed randomized algorithms of time $10.88^{rk} n^{O(1)}$

and space $O(2^{rk} + rn)$, and deterministic algorithms of time $2^{O(rk)}n^{O(1)}$ and space $O(2^{rk} + rn)$ for the problems. The deterministic upper bound was further improved to $2^{5rk-4k}\binom{6(r-1)k+k}{rk}n^{O(1)}$ (still with exponential space) by Fellows et al. [7]. The problems of packing a small subgraph in a given graph, such as the triangle packing problem, can be transformed into the P-MAX r-SET PACKING problem directly. Algorithms for this kind of graph packing problems have also been studied [7, 15, 20].

Currently, the best randomized and deterministic algorithms for the P-MAX PATH, P-MAX r-D MATCHING, and P-MAX r-SET PACKING problems [2, 7, 13] are based on the *color-coding* technique developed by Alon, Yuster, and Zwick [2]. The technique is based on constructing an (n, k)-family of perfect hash functions and a dynamic programming process on k-colored instances. For example, the randomized algorithm for the P-MAX PATH problem given in [2] based on this technique runs in time $(2e)^k n^{O(1)} = 5.44^k n^{O(1)}$ and space $2^k n^{O(1)}$. Because of the lower bound $\Omega(e^k)$ on the size of (n, k)-families of perfect hash functions [18], and of the nature of dynamic programming, it seems difficult to further improve the time complexity and to avoid exponential space complexity for algorithms based on this technique.

1.3. Our contributions. In this paper, we develop new exponential-time algorithmic techniques that lead to improved randomized and deterministic algorithms for the P-MAX PATH, P-MAX r-D MATCHING, and P-MAX r-SET PACKING problems. Our main idea is using a divide-and-conquer method, as follows. Suppose that we are looking for a subset S_h of h elements in a universal set U. Moreover, we assume that the subset S_h is "hierarchical" in the sense that such a subset is constituted by smaller subsets of similar properties. We first randomly partition the universal set U into two disjoint parts, U_1 and U_2. A simple probabilistic analysis shows that with an effective probability, the desired subset S_h is evenly split by this random partition into two smaller subsets of similar properties. This enables us to recursively look in each of the parts U_1 and U_2 for a smaller subset of $h/2$ elements in S_h and of similar properties, and to finally combine the smaller subsets to obtain the desired subset S_h in the original universal set U.

Take the P-MAX 3-SET PACKING problem as an example. For a given collection $\mathcal{C}$ of 3-sets and a parameter k, the universal set U is the set of all elements that appear in the 3-sets in $\mathcal{C}$, and the desired subset S_{3k} consists of the $3k$ elements from k 3-sets that make a k-packing $\mathcal{P}_k$ of the maximum weight in $\mathcal{C}$. It is not difficult to see that with a probability of at least $1/2^{3k}$, a random partition of U into two parts U_1 and U_2 includes all $3k/2$ elements from $k/2$ 3-sets in $\mathcal{P}_k$ in the part U_1 and all $3k/2$ elements from the other $k/2$ 3-sets in $\mathcal{P}_k$ in the part U_2.[1] Moreover, the two parts U_1 and U_2 induce two subcollections $\mathcal{C}_1$ and $\mathcal{C}_2$ of $\mathcal{C}$, respectively, after deleting all 3-sets that contain both elements in U_1 and U_2. Therefore, a k-packing of the maximum weight in $\mathcal{C}$ can be constructed by combining a $(k/2)$-packing of the maximum weight in $\mathcal{C}_1$ and a $(k/2)$-packing of the maximum weight in $\mathcal{C}_2$, which can be recursively constructed.

This simple method leads directly to randomized algorithms with improved running time and with polynomial space. For the P-MAX PATH problem, this new method gives a randomized algorithm of time $O(4^k k^{2.7} m)$ and space $O(nk \log k + m)$, improving the previously best randomized algorithm for the problem of time $O(5.44^k km)$ and space $O(2^k kn + m)$ [2]. For the P-MAX r-D MATCHING and P-MAX r-SET PACKING problems, the method gives randomized algorithms of time $4^{(r-1)k}k^{\log k/3}n^{O(1)}$ and space $(rk \log k + rn)$, improving the previously best randomized algorithms for

[1] We will show in section 4 that with a more careful analysis, we can derive a better probability.

the problems of time $10.88^{rk}n^{O(1)}$ and space $O(2^{rk} + rn)$ [13]. Furthermore, these randomized algorithms can be derandomized, which leads to deterministic algorithms that significantly improve previous best deterministic algorithms for the corresponding problems.

The major techniques and results reported in this paper were independently discovered by the research group at Texas A&M University and by the research group at RWTH Aachen University. Preliminary results from the two groups were reported independently at SODA'2007 and WG'2006, respectively [5, 12]. The current paper is a result of collaboration between the two groups, which enhances, extends, and refines the results reported in [5, 12].

2. On a class of recurrence relations. The analysis of many of our randomized and deterministic algorithms presented in this paper requires solving recurrence relations of a special form, which seems neither standard nor trivial. This section is devoted to giving a thorough and formal study of this class of recurrence relations.

THEOREM 2.1. *Let $T(k,n)$ be a function satisfying the following conditions: there are functions $t(n)$, $f(k)$, and $h(n)$, and a real number $a \geq 1$ such that*

(A) *$T(k,n) \leq O(a^{2k}t(n))$ for all $k \leq h(n)$; and*

(B) *for $k > h(n)$, the function $T(k,n)$ satisfies*

$$T(k,n) \leq f(k)a^k[t(n) + T(k_1,n) + T(k_2,n)],$$

where $k_1 = \lceil k/2 \rceil$ and $k_2 = \lfloor k/2 \rfloor$.

Then $T(k,n) = O(a^{2k}g(k)t(n))$, where $g(k)$ is any function satisfying:

(C) *$g(k) \geq 1$ for all $k \geq 1$; and*

(D) *there exist an integer $k_0 \geq 1$ and a positive real number $d_0 < 1$ such that for all $k \geq k_0$,*

$$H(a,k,f(k),g(k)) \stackrel{\text{def}}{=\!=} \frac{a^{2k_1}f(k)g(k_1) + a^{2k_2}f(k)g(k_2)}{a^k g(k)} \leq d_0.$$

Proof. Pick a constant c_0 so that $d_0(1 + 1/(2c_0)) \leq 1$. We claim

$$T(k,n) \leq c_0 a^{2k} g(k)t(n), \tag{1}$$

which will prove the theorem.

We prove (1) by induction on k. For the values of k that are bounded by $h(n)$, (1) holds true by condition (A) if the constant c_0 is sufficiently large. Applying induction on condition (B) when $k > h(n)$, we obtain

$$\begin{aligned} T(k,n) &\leq f(k)a^k[t(n) + T(k_1,n) + T(k_2,n)] \\ &\leq f(k)a^k[t(n) + c_0 a^{2k_1}g(k_1)t(n) + c_0 a^{2k_2}g(k_2)t(n)] \\ &= c_0 a^{2k} g(k)t(n)\left[\frac{f(k)}{c_0 a^k g(k)} + \frac{a^{2k_1}f(k)g(k_1) + a^{2k_2}f(k)g(k_2)}{a^k g(k)}\right]. \end{aligned} \tag{2}$$

By condition (D), the second term in the bracket in (2) is $H(a,k,f(k),g(k))$, which is bounded by d_0. Combining condition (D) with the fact that the values k_1, k_2, $g(k_1)$, $g(k_2)$, a, and k are all larger than or equal to 1, we also get the following bound for the first term in the bracket in (2):

$$\frac{f(k)}{c_0 a^k g(k)} \leq \frac{d_0}{2c_0}.$$

By the way we selected the value of c_0, we have $d_0 + d_0/(2c_0) = d_0(1 + 1/(2c_0)) \leq 1$. Therefore,

$$T(k, n) \leq c_0 a^{2k} g(k) t(n),$$

and the induction goes through. □

Applying Theorem 2.1, we obtain a sequence of corollaries that give the bounds that are specifically needed in the analysis of our algorithms presented in this paper. Corollary 2.2 below will be used in the analysis of our algorithm for the P-MAX PATH problem.

COROLLARY 2.2. *Suppose that a function $T(k, n)$ satisfies $T(1, n) = O(t(n))$ and the following recurrence relation for $k \geq 2$:*

$$T(k, n) \leq c_0 a^k [t(n) + T(k_1, n) + T(k_2, n)],$$

where $k_1 = \lceil k/2 \rceil$, $k_2 = \lfloor k/2 \rfloor$, and c_0 and $a \geq 1$ are constants. Then $T(k, n) = O(a^{2k} k^\alpha t(n))$, where α is any constant satisfying $\alpha > \log_2(c_0(a^2 + 1)/a)$.

Proof. Using the notation in Theorem 2.1, here we have $f(k) = c_0$ and $h(n) \equiv 1$. We verify that the function $g(k) = k^\alpha$ satisfies conditions (C) and (D) in Theorem 2.1. Condition (C) is trivially satisfied. To verify condition (D), we have

$$H(a, k, c_0, g(k)) = \frac{c_0 a^{2k_1} k_1^\alpha + c_0 a^{2k_2} k_2^\alpha}{a^k k^\alpha}.$$

If k is even, we have

$$H(a, k, c_0, g(k)) = \frac{c_0 a^k (k/2)^\alpha + c_0 a^k (k/2)^\alpha}{a^k k^\alpha} = \frac{c_0}{2^{\alpha-1}}.$$

Since $\alpha > \log_2(c_0(a^2 + 1)/a)$ and it is easy to see that $(a^2 + 1)/a \geq 2$, we get $c_0/2^{\alpha-1} < 1$.

On the other hand, suppose that k is odd; then

$$\begin{aligned} H(a, k, c_0, g(k)) &= \frac{c_0 a^{k+1}((k+1)/2)^\alpha + c_0 a^{k-1}((k-1)/2)^\alpha}{a^k k^\alpha} \\ &= \frac{c_0}{2^\alpha a} \left[a^2 \left(1 + \frac{1}{k}\right)^\alpha + \left(1 - \frac{1}{k}\right)^\alpha \right]. \end{aligned}$$

Since when $k \to \infty$, $a^2(1 + 1/k)^\alpha + (1 - 1/k)^\alpha \to a^2 + 1$, the above value approaches $c_0(a^2+1)/(2^\alpha a)$. From $\alpha > \log_2(c_0(a^2+1)/a)$ we have $c_0(a^2+1)/(2^\alpha a) < 1$. Therefore, there must be a constant k_0 such that $d = c_0[a^2(1 + 1/k_0)^\alpha + (1 - 1/k_0)^\alpha]/(2^\alpha a) < 1$, and for all odd numbers $k \geq k_0$, we have $H(a, k, c_0, g(k)) \leq d$.

Now if we let $d_0 = \max\{c_0/2^{\alpha-1}, d\}$, then $d_0 < 1$ and for all $k \geq k_0$ we have

$$H(a, k, c_0, g(k)) \leq d_0.$$

Thus, the function $g(k) = k^\alpha$ satisfies condition (D) in Theorem 2.1. The corollary follows. □

Corollary 2.3 below will be used in the analysis of our algorithms for the P-MAX r-D MATCHING and P-MAX r-SET PACKING problems.

COROLLARY 2.3. *Suppose that a function $T(k, n)$ satisfies $T(1, n) = O(t(n))$ and the following recurrence relation for $k \geq 2$:*

$$T(k, n) \leq c_0 k^b a^k [t(n) + T(k_1, n) + T(k_2, n)],$$

where $k_1 = \lceil k/2 \rceil$, $k_2 = \lfloor k/2 \rfloor$, *and* c_0, $b > 0$, *and* $a \geq 1$ *are all constants. Then* $T(k,n) = O(a^{2k} k^{\alpha \log k} t(n))$, *where* α *is any constant satisfying* $\alpha > b/2$.

Proof. Using the notation in Theorem 2.1, here we have $f(k) = c_0 k^b$ and $h(n) \equiv 1$. We verify that the function $g(k) = k^{\alpha \log k}$ satisfies conditions (C) and (D) in Theorem 2.1. Condition (C) is trivially satisfied. To verify condition (D), we have

$$\begin{aligned} H(a,k,c_0k^b,g(k)) &= \frac{c_0k^b a^{2k_1} k_1^{\alpha \log k_1} + c_0 k^b a^{2k_2} k_2^{\alpha \log k_2}}{a^k k^{\alpha \log k}} \\ &\leq \frac{c_0 k^b a^{k+1} ((k+1)/2)^{\alpha \log((k+1)/2)} + c_0 k^b a^{k+1} ((k+1)/2)^{\alpha \log((k+1)/2)}}{a^k k^{\alpha \log k}} \\ &= \frac{2^{\alpha+1} c_0 k^b a}{(k+1)^{2\alpha}} \cdot \frac{(k+1)^{\alpha \log(k+1)}}{k^{\alpha \log k}} \\ &\leq \frac{2^{\alpha+1} c_0 a}{(k+1)^{2\alpha-b}} \cdot \frac{(k+1)^{\alpha \log(k+1)}}{k^{\alpha \log k}}. \end{aligned}$$

To see the limit of this value when k approaches ∞, note that if we let $r > 1$ be a constant such that $2\alpha - b - 2\alpha \log r > 0$ (note $\alpha > b/2$), then the above expression can be rewritten as

$$\frac{2^{\alpha+1} c_0 a}{(k+1)^{2\alpha-b}} \cdot \frac{(k+1)^{\alpha \log(k+1)}}{k^{\alpha \log k}} = \frac{2^{\alpha+1} c_0 a}{r^{\alpha \log r} (k+1)^{2\alpha-b-2\alpha \log r}} \cdot \frac{((k+1)/r)^{\alpha \log((k+1)/r)}}{k^{\alpha \log k}}.$$

Now since $r > 1$, $(k+1)/r < k$ when k is sufficiently large. Therefore, the value approaches 0 when $k \to \infty$. In particular, this implies that there is an integer k_0 and a constant $d_0 < 1$ such that when $k \geq k_0$, $H(a,k,c_0k^b,g(k)) \leq d_0$. This completes the proof of the corollary. □

3. A randomized algorithm for the PATH problem. Now we are ready to present our randomized algorithms. The first problem we are dealing with is the P-MAX PATH problem that looks for a k-path of the maximum weight in a weighted graph.

Fix a weighted graph $G = (V, E)$. For any $V' \subseteq V$, denote by $G[V']$ the subgraph of G induced by V'. The *concatenation* of two paths $\rho_1 = \langle v_1, \ldots, v_l \rangle$ and $\rho_2 = \langle w_1, \ldots, w_h \rangle$ in G, where $[v_l, w_1]$ is an edge in G, is the path $\langle v_1, \ldots, v_l, w_1 \ldots, w_h \rangle$. We denote by $\rho_\emptyset$ the special 0-*path* (i.e., the empty path containing no vertex) and define that the concatenation of $\rho_\emptyset$ and any path ρ gives the path ρ. An h-path ρ is also called a (v,h)-*path* if v is an end vertex of ρ.

Let P_l be a set of l-paths in G, and let $V' \subseteq V$ such that no vertex in V' is on any path in P_l. A (v,h)-*path* ρ *is in* $P_l \odot V'$ if $v \in V'$ and ρ is a concatenation of an l-path in P_l and an $(h-l)$-path in $G[V']$. In particular, for $P_0 = \{\rho_\emptyset\}$, any $(v,1)$-path in $P_0 \odot V'$ consists of the single vertex v in V'.

On a set P_l of l-paths in G and $V' \subseteq V$, where P_l contains at most one (v,l)-path for each vertex v, and no vertex in V' is on any path in P_l, our algorithm **find-paths**(P_l, V', h) returns a set P_{l+h} of $(l+h)$-paths in $P_l \odot V'$. In particular, the algorithm **find-paths**$(\{\rho_\emptyset\}, V, k)$ returns a set of k-paths in the graph G. The algorithm is given in Figure 1.

THEOREM 3.1. *Let* P_l *and* V' *be defined as above. For any vertex* v *in* V', *if there are* $(v, l+h)$*-paths in* $P_l \odot V'$, *then with probability larger than* $1 - 1/e > 0.632$ *(here* e *is the base of the natural logarithm), the set* P_{l+h} *returned by the algorithm* **find-paths**(P_l, V', h) *contains a* $(v, l+h)$*-path in* $P_l \odot V'$ *whose weight is the maximum*

find-paths(P_l, V', h)
input: a set P_l of l-paths; $V' \subseteq V$ and no vertex in V' is on a path in P_l; an integer $h \geq 1$;
output: a set P_{l+h} of $(l+h)$-paths in $P_l \odot V'$;
1. $P_{l+h} = \emptyset$;
2. **if** $h = 1$ **then**
2.1. **if** $P_l = \{\rho_\emptyset\}$ **then** P_{l+1} contains a $(u, 1)$-path for each vertex $u \in V'$; **return** P_{l+1};
2.2. **else for** each (w, l)-path ρ_l in P_l and each $u \in V'$, where $[w, u]$ is an edge in G, **do**
2.3. concatenate ρ_l and u to make a $(u, l+1)$-path ρ_{l+1} in $P_l \odot V'$;
2.4. **if** P_{l+1} contains no $(u, l+1)$-path **then** add ρ_{l+1} to P_{l+1};
2.5. **else if** the $(u, l+1)$-path ρ'_{l+1} in P_{l+1} has a weight smaller than that of ρ_{l+1}
2.6. **then** replace ρ'_{l+1} in P_{l+1} by ρ_{l+1};
2.7. **return** P_{l+1};
3. **loop** $2.51 \cdot 2^h$ times **do**
3.1. randomly partition the vertices in V' into two parts V_L and V_R;
3.2. $P^L_{l+\lceil h/2 \rceil} =$ **find-paths**$(P_l, V_L, \lceil h/2 \rceil)$;
3.3. **if** $P^L_{l+\lceil h/2 \rceil} \neq \emptyset$ **then**
3.4. $P^R_{l+h} =$ **find-paths**$(P^L_{l+\lceil h/2 \rceil}, V_R, \lfloor h/2 \rfloor)$;
3.5. **for** each $(u, l+h)$-path ρ_{l+h} in P^R_{l+h} **do**
3.6. **if** P_{l+h} contains no $(u, l+h)$-path in $P_l \odot V'$ **then** add ρ_{l+h} to P_{l+h};
3.7. **else if** the $(u, l+h)$-path ρ'_{l+h} in P_{l+h} has a weight smaller than that of ρ_{l+h}
3.8. **then** replace ρ'_{l+h} in P_{l+h} by ρ_{l+h};
4. **return** P_{l+h}.

FIG. 1. *A randomized divide-and-conquer algorithm for* P-MAX PATH.

over all $(v, l+h)$-paths in $P_l \odot V'$. The algorithm **find-paths**(P_l, V', h) *runs in time $O(4^h h^{2.7} m)$ and in space $O(n(l+h)\log h + m)$.*

Proof. First note that by steps 2.4–2.6 and steps 3.6–3.8, if the set P_{l+h} contains a $(v, l+h)$-path ρ, then ρ must be a valid $(v, l+h)$-path in $P_l \odot V'$. Therefore, if there is no $(v, l+h)$-path in $P_l \odot V'$, then the set P_{l+h} cannot contain a $(v, l+h)$-path.

Thus we assume that in the graph G there are $(v, l+h)$-paths in $P_l \odot V'$. Let

$$\rho_{l+h} = \langle u_1, \ldots, u_l, w_1, \ldots, w_h \rangle$$

be a $(v, l+h)$-path in $P_l \odot V'$ whose weight is the maximum over all $(v, l+h)$-paths in $P_l \odot V'$, where $\langle u_1, \ldots, u_l \rangle$ is an l-path in P_l, $\langle w_1, \ldots, w_h \rangle$ is an h-path in $G[V']$, and $w_h = v$. We prove the theorem by induction on $h \geq 1$.

Consider the case $h = 1$. If $P_l = \{\rho_\emptyset\}$ (in this case $l = 0$), then the set P_{l+1} returned by step 2.1 contains the (unique) $(v, 1)$-path in $P_l \odot V'$, which is obviously of the maximum weight. On the other hand, if $l > 0$, then when the (u_l, l)-path $\langle u_1, \ldots, u_l \rangle$ in P_l and the vertex $w_h = w_1 = v$ are examined in step 2.2, the path ρ_{l+1} is constructed in step 2.3, and steps 2.4–2.6 ensure that a $(v, l+1)$-path of the maximum weight is included in the set P_{l+1}. This proves the case $h = 1$.

Now suppose that $h > 1$. We rewrite the path ρ_{l+h} as

$$\rho_{l+h} = \langle u_1, \ldots, u_l, w_1, \ldots, w_{h_1}, \ldots, w_h \rangle,$$

where $h_1 = \lceil h/2 \rceil$. With probability $1/2^h$, step 3.1 of the algorithm puts vertices $w_1, \ldots, w_{h_1}$ into V_L, and vertices $w_{h_1+1}, \ldots, w_h$ into V_R. If this is the case, then the path

$$\rho_{l+h_1} = \langle u_1, \ldots, u_l, w_1, \ldots, w_{h_1} \rangle$$

is a $(w_{h_1}, l+h_1)$-path in $P_l \odot V_L$. By the induction hypothesis, with probability larger than $1 - 1/e$, the set $P^L_{l+h_1}$ obtained in step 3.2 contains a $(w_{h_1}, l+h_1)$-path $\overline{\rho}_{l+h_1}$

in $P_l \odot V_L$ whose weight is at least as large as that of ρ_{l+h_1}. Now the concatenation of the path $\overline{\rho}_{l+h_1}$ and the path $\langle w_{h_1+1}, \ldots, w_h \rangle$ is a $(w_h, (l+h_1)+(h-h_1))$-path (i.e., a $(v, l+h)$-path) in $P^L_{l+h_1} \odot V_R$. Thus, by our induction hypothesis again, with probability larger than $1-1/e$, the set P^R_{l+h} obtained in step 3.4 contains a $(v, l+h)$-path $\overline{\rho}_{l+h}$ in $P^L_{l+h_1} \odot V_R$ whose weight is at least as large as the sum of the weight of the path $\overline{\rho}_{l+h_1}$ and the weight of the path $\langle w_{h_1+1}, \ldots, w_h \rangle$. Since the weight of $\overline{\rho}_{l+h_1}$ is not smaller than that of ρ_{l+h_1}, we conclude that the weight of the path $\overline{\rho}_{l+h}$ is not smaller than that of ρ_{l+h}. Finally, since the path $\overline{\rho}_{l+h}$ is a concatenation of a $(w, l+h_1)$-path ρ'_{l+h_1} in $P_l \odot V_L$ (for some vertex $w \in V_L$) and a path in $G[V_R]$, where the path ρ'_{l+h_1} is a concatenation of a path in P_l and a path in $G[V_L]$, we derive that $\overline{\rho}_{l+h}$ is actually a $(v, l+h)$-path in $P_l \odot V'$. Since the weight of $\overline{\rho}_{l+h}$ is not smaller than the weight of ρ_{l+h}, and by our assumption, the path ρ_{l+h} has the maximum weight over all $(v, l+h)$-paths in $P_l \odot V'$, we conclude that $\overline{\rho}_{l+h}$ must also be a $(v, l+h)$-path of the maximum weight in $P_l \odot V'$.

In conclusion, with probability $1/2^h$, step 3.1 includes the vertices $w_1, \ldots, w_{h_1}$ in V_L and the vertices $w_{h_1+1}, \ldots, w_h$ in V_R. If this is the partition, then with probability larger than $1-1/e$, the set $P^L_{l+h_1}$ in step 3.2 contains the $(w_{h_1}, l+h_1)$-path $\overline{\rho}_{l+h_1}$. In case the set $P^L_{l+h_1}$ contains the path $\overline{\rho}_{l+h_1}$, with probability larger than $1-1/e$, the set P^R_{l+h} in step 3.4 contains a $(v, l+h)$-path of the maximum weight. Therefore, by steps 3.5–3.8, in each loop of step 3, the probability q that the set P_{l+h} contains a $(v, l+h)$-path of the maximum weight is larger than

$$\frac{(1-1/e)^2}{2^h} > \frac{1}{2.51 \cdot 2^h}.$$

Since step 3 of the algorithm loops $2.51 \cdot 2^h$ times, the overall probability that the algorithm returns the set P_{l+h} that contains a $(v, l+h)$-path of the maximum weight is

$$1-(1-q)^{2.51 \cdot 2^h} > 1-\left(1-\frac{1}{2.51 \cdot 2^h}\right)^{2.51 \cdot 2^h} > 1-\frac{1}{e}.$$

This proves the first part of the theorem.

To analyze the time complexity, let $T(h, m)$ be the running time of the algorithm **find-paths**(P_l, V', h), where m is the number of edges in the original graph G. Clearly we have $T(1, m) = O(m)$. From the algorithm, we have the following recurrence relation when $h > 1$:

$$T(h, m) = 2.51 \cdot 2^h [cm + T(\lceil h/2 \rceil, m) + T(\lfloor h/2 \rfloor, m)],$$

where $c > 0$ is a constant. Using the notation in Corollary 2.2, here we have $c_0 = 2.51$, $a = 2$, and $t(n) = cm$. Now $\log_2(c_0(a^2+1)/a) \geq 2.64$. Thus, by Corollary 2.2, the running time $T(h, m)$ of the algorithm **find-paths**(P_l, V', h) is $O(4^h h^{2.7} m)$.

In terms of the space complexity, each recursive call to the algorithm **find-paths**(P_l, V', h) uses $O(n(l+h))$ space (mainly for the sets $P^L_{l+h_1}$, P^R_{l+h}, and P_{l+h}, noting that for each vertex w in the graph G, each of these sets contains at most one $(w, *)$-path). Since the recursive depth of the algorithm **find-paths**(P_l, V', h) is $O(\log h)$, we conclude that the space complexity of the algorithm **find-paths**(P_l, V', h) is $O(n(l+h)\log h + m)$. □

To obtain a randomized algorithm solving the P-MAX PATH problem with a required error bound, we simply run the algorithm **find-paths**$(\{\rho_\emptyset\}, V, k)$ sufficiently

many times, each taking $O(4^k k^{2.7} m)$ time and $O(nk \log k + m)$ space. For example, to achieve an error bound of 0.0001, we can run the algorithm t times, where t satisfies $(1/e)^t \leq 0.0001$ (e.g., $t = 10$). Note that the set P_k returned by the algorithm **find-paths**$(\{\rho_\emptyset\}, V, k)$ contains at most one (v, k)-path for each vertex v in the graph G. Therefore, by picking the (v, k)-path of the maximum weight in P_k, this process returns a k-path of the maximum weight in the graph G with an arbitrarily small error bound.

COROLLARY 3.2. *There is a randomized algorithm of time $O(4^k k^{2.7} m)$ and space $O(nk \log k + m)$ that solves the* P-MAX PATH *problem with an arbitrarily small error bound.*

Remark. The algorithm **find-paths** can be used directly to solve the P-MAX PATH problem on *directed graphs*, as long as we interpret the edge $[w, u]$ in step 2.2 of the algorithm as a directed edge from w to u. The proof of Theorem 3.1 can be applied to directed graphs with no change.

We compare our algorithm in Corollary 3.2 with previously known algorithms for the P-MAX PATH problem. To our knowledge, there are two kinds of randomized algorithms for the problem. The first kind is based on random permutations of vertices followed by searching in a directed acyclic graph [2, 11]. The algorithm runs in time $O(mk!)$ and space $O(m)$. The second kind, proposed by Alon, Yuster, and Zwick [2], is based on random coloring of vertices in a graph followed by dynamic programming to search for a simple path of length k in the colored graph. The algorithm runs in time $O((2e)^k km) = O(5.44^k km)$ and space $O(2^k kn + m)$ (the space is mainly for the dynamic programming phase). Compared to these algorithms, our algorithm has a significantly improved running time and uses polynomial space. In fact, if we are interested only in knowing whether the graph has a path of length k, a slight modification of our algorithm can further reduce the space complexity to $O(n \log k + m)$.

4. Randomized algorithms for the MATCHING and PACKING problems. The randomized divide-and-conquer method described in the previous section can also be used to develop improved algorithms for MATCHING and PACKING problems.

Recall that any algorithm solving the P-MAX r-SET PACKING problem can be used directly to solve the P-MAX r-D MATCHING problem. Therefore, our discussion in this section will be focused on the P-MAX r-SET PACKING problem, which looks for a k-packing of the maximum weight in a collection of r-sets. We have the following result.

THEOREM 4.1. *There is a randomized algorithm that solves the* P-MAX r-SET PACKING *problem in time $O(4^{(r-1)k} k^{\log k/3} rn)$ and space $O(rk \log k + rn)$, where n is the number of r-sets in the given instance of the* P-MAX r-SET PACKING *problem.*

Proof. Consider the algorithm in Figure 2. Let $k_1 = \lceil k/2 \rceil$. We first prove, by induction on the parameter k, that if the collection $\mathcal{C}$ contains k-packings, then with probability larger than $1 - 1/e$, the algorithm **set-packing**$(\mathcal{C}, k)$ returns a k-packing in $\mathcal{C}$ whose weight is the maximum over all k-packings in $\mathcal{C}$.

This is obviously true when $k = 1$. Now suppose that $k > 1$. Again first note that if the collection $\mathcal{C}$ has no k-packings, then the algorithm **set-packing**$(\mathcal{C}, k)$ must return the empty set $\emptyset$. Now suppose that $\mathcal{C}$ contains k-packings, and let $\mathcal{P}_k$ be a k-packing in $\mathcal{C}$ whose weight is the maximum over all k-packings in $\mathcal{C}$. With a probability $\binom{k}{k_1}/2^{rk}$, step 3.1 of the algorithm partitions the rk elements in the k r-sets in $\mathcal{P}_k$ such that the rk_1 elements in (any) k_1 r-sets in $\mathcal{P}_k$ are in S_L while the $r(k - k_1)$ elements in the other $k - k_1$ r-sets in $\mathcal{P}_k$ are in S_R. If this is the case, let $\mathcal{P}_{k_1}^L$ be the

set-packing$(\mathcal{C}, k)$
input: a collection $\mathcal{C}$ of r-sets, in which each r-set is assigned a weight, and an integer $k \geq 1$;
output: a k-packing in $\mathcal{C}$ that has the maximum weight over all k-packings in $\mathcal{C}$,
or $\emptyset$ if $\mathcal{C}$ has no k-packing.

1. **if** $k = 1$ **then**
   if $\mathcal{C} \neq \emptyset$ **then** **return** an r-set of the maximum weight in $\mathcal{C}$ **else return** $\emptyset$;
2. $\mathcal{P} = \emptyset$;
3. **loop** $2.51 \cdot 2^{rk}/\binom{k}{\lceil k/2 \rceil}$ times **do**
3.1. randomly partition the set elements into two parts S_L and S_R;
3.2. let $\mathcal{C}_L$ be the subcollection of r-sets in $\mathcal{C}$ in which all elements are in S_L;
3.3. let $\mathcal{C}_R$ be the subcollection of r-sets in $\mathcal{C}$ in which all elements are in S_R;
3.4. $\mathcal{P}_L =$ **set-packing**$(\mathcal{C}_L, \lceil k/2 \rceil)$; $\mathcal{P}_R =$ **set-packing**$(\mathcal{C}_R, \lfloor k/2 \rfloor)$;
3.5. **if** $\mathcal{P}_L \neq \emptyset$, $\mathcal{P}_R \neq \emptyset$, and the sum of the weights of $\mathcal{P}_L$ and $\mathcal{P}_R$ is larger than the weight of $\mathcal{P}$
then $\mathcal{P} = \mathcal{P}_L \cup \mathcal{P}_R$;
4. **return** $\mathcal{P}$.

FIG. 2. *A randomized divide-and-conquer algorithm for* P-MAX r-SET PACKING.

set of k_1 r-sets in $\mathcal{P}_k$ whose elements are all in S_L and let $\mathcal{P}^R_{k-k_1}$ be the set of $k - k_1$ r-sets in $\mathcal{P}_k$ whose elements are all in S_R. Note that $\mathcal{P}^L_{k_1}$ is a k_1-packing in $\mathcal{C}_L$ and that $\mathcal{P}^R_{k-k_1}$ is a $(k - k_1)$-packing in $\mathcal{C}_R$. Thus, by the induction hypothesis, with a probability larger than $(1 - 1/e)^2$, step 3.4 of the algorithm generates a k_1-packing $\mathcal{P}_L$ in $\mathcal{C}_L$ and a $(k - k_1)$-packing $\mathcal{P}_R$ in $\mathcal{C}_R$, such that the weight of $\mathcal{P}_L$ is not smaller than that of $\mathcal{P}^L_{k_1}$ and that the weight of $\mathcal{P}_R$ is not smaller than that of $\mathcal{P}^R_{k-k_1}$. Note that the union of $\mathcal{P}_L$ and $\mathcal{P}_R$ must be a k-packing in $\mathcal{C}$ because no set element appears in both $\mathcal{C}_L$ and $\mathcal{C}_R$. Since the union of $\mathcal{P}^L_{k_1}$ and $\mathcal{P}^R_{k-k_1}$ is the k-packing $\mathcal{P}_k$ that has the maximum weight over all k-packings in $\mathcal{C}$, we conclude that the union of $\mathcal{P}_L$ and $\mathcal{P}_R$ must be a k-packing of the maximum weight in $\mathcal{C}$. In summary, if the collection $\mathcal{C}$ contains k-packings, then with a probability q larger than

$$\frac{(1 - 1/e)^2 \binom{k}{k_1}}{2^{rk}} > \frac{\binom{k}{k_1}}{2.51 \cdot 2^{rk}},$$

an execution of steps 3.1–3.5 of the algorithm will make $\mathcal{P}$ a k-packing of the maximum weight in $\mathcal{C}$. Now since the loop in step 3 is executed $2.51 \cdot 2^{rk}/\binom{k}{k_1}$ times, with probability at least

$$1 - (1 - q)^{2.51 \cdot 2^{rk}/\binom{k}{k_1}} > 1 - \left(1 - \frac{\binom{k}{k_1}}{2.51 \cdot 2^{rk}}\right)^{2.51 \cdot 2^{rk}/\binom{k}{k_1}} > 1 - \frac{1}{e},$$

the algorithm **set-packing**$(\mathcal{C}, k)$ returns a k-packing of the maximum weight in $\mathcal{C}$.

To analyze the complexity of the algorithm, let $T(k, n)$ be the running time of the algorithm **set-packing**$(\mathcal{C}, k)$, where n is the total number of r-sets in the collection $\mathcal{C}$. Clearly we have $T(1, n) = O(rn)$. Moreover, for $k > 1$,

$$\begin{aligned} T(k, n) &= \left(\frac{2.51 \cdot 2^{rk}}{\binom{k}{k_1}}\right) \cdot [crn + T(k_1, n) + T(k - k_1, n)] \\ &\leq 14\sqrt{\pi}\sqrt{k}(2^{r-1})^k [crn + T(\lceil k/2 \rceil, n) + T(\lfloor k/2 \rfloor, n)], \end{aligned}$$

where c is a constant, and based on Stirling's formula [9], we have used the inequality $\binom{k}{k_1} \geq 2^k/(2e\sqrt{\pi k})$. Using the notation in Corollary 2.3, here we have $c_0 = 14\sqrt{\pi}$, $k^b = \sqrt{k} = k^{1/2}$, $a = 2^{r-1}$, and $t(n) = crn$. By Corollary 2.3, the running time of the

algorithm **set-packing**$(\mathcal{C}, k)$ is bounded by

$$T(k,n) = O((2^{r-1})^{2k} k^{\log k/3} rn) = O(4^{(r-1)k} k^{\log k/3} rn).$$

Moreover, since the recursion depth of the algorithm is bounded by $O(\log k)$, it is easy to see that the space complexity of the algorithm is $O(rk \log k + rn)$.

The theorem now follows by an argument similar to that for Corollary 3.2. ∎

COROLLARY 4.2. *The* P-MAX r-D MATCHING *problem can be solved in time* $O(4^{(r-1)k} k^{\log k/3} rn)$ *and space* $O(rk \log k + rn)$*, where* n *is the number of points in the given instance of the* P-MAX r-D MATCHING *problem.*

The previously best randomized algorithms for the P-MAX r-D MATCHING and P-MAX r-SET PACKING problems are due to Koutis and have running time $10.88^{rk} n^{O(1)}$ and space complexity $O(2^{rk} + rn)$. Therefore, Theorem 4.1 and Corollary 4.2 not only give significantly improved running time but also bring the space complexity from exponential down to polynomial.

The techniques used in Theorem 4.1 can be used to solve graph packing problems in a very general form. Let H be a fixed graph. A *k-H-packing* of a graph G is a collection of k vertex-disjoint subgraphs $\mathcal{P}_k = \{H_1, \ldots, H_k\}$ of G such that each H_i is isomorphic to the graph H. Suppose that there is also a *weight function* f_W from the subgraphs of G to real numbers (that is, for each subgraph G' of G, $f_W(G')$ is a real number that is the *weight* of the subgraph G'). Then we define the *weight* of a k-H-packing $\mathcal{P}_k$ to be the sum of the weights of the subgraphs in $\mathcal{P}_k$. Now we can define a graph packing problem as follows.

> P-MAX H-GRAPH PACKING: Given a graph G and a parameter k, where there is a weight function f_W from the subgraphs of G to real numbers, either construct a k-H-packing of G that has the maximum weight over all k-H-packings of G, or report that no k-H-packing exists in G.

Suppose that the graph H contains r vertices. Then the P-MAX H-GRAPH PACKING problem can be reduced to the P-MAX r-SET PACKING problem, as follows. On the input graph G, let $\mathcal{C}_G$ be the collection of all subsets V_r of r vertices in G such that the induced subgraph $G[V_r]$ contains the graph H (or more formally, H is isomorphic to a subgraph of $G[V_r]$). For each subset V_r in $\mathcal{C}_G$, define the weight of V_r to be the weight of H', where H' is the subgraph in $G[V_r]$ that is isomorphic to H, and the weight of H' is the maximum over all subgraphs of $G[V_r]$ that are isomorphic to H. It is easy to verify that there is a one-to-one mapping between the k-packings of the maximum weight in the collection $\mathcal{C}_G$ and the k-H-packings of the maximum weight of the graph G. Thus, the P-MAX H-GRAPH PACKING problem on the graph G can be solved directly by applying the algorithm given in Theorem 4.1 to the collection $\mathcal{C}_G$. Furthermore, we can avoid the explicit construction of the collection $\mathcal{C}_G$ and further reduce the complexity of the algorithm. The detailed algorithm is given in Figure 3.

THEOREM 4.3. *Let* H *be a fixed graph of* r *vertices. Then the algorithm* H**-graph packing** *solves the* P-MAX H-GRAPH PACKING *problem in time* $O(4^{(r-1)k} k^{\log k/3} n^r r^2)$ *and space* $O(rk \log k + n^2)$ *on a graph of* n *vertices.*

Proof. The correctness proof and the complexity analysis of the algorithm H**-graph packing** are completely similar to that of Theorem 4.1, except for the case $k = 1$ in step 1. To find the subgraph H' in G that is isomorphic to H and has the maximum weight, we enumerate all subsets of r vertices in G. For each subset V_r, we consider all possible vertex mappings from H to $G[V_r]$. Note that each isomorphic mapping from H to a subgraph of G is uniquely determined by a subset V_r of r vertices

H-graph packing(G, k)
input: a graph G, an integer $k \geq 1$, and a weight function f_W from subgraphs of G to real numbers;
output: a k-H-packing of G that has the maximum weight over all k-H-packings of G,
or $\emptyset$ if there is no k-H-packing of G.

1. **if** $k = 1$ **then**
   if H is isomorphic to a subgraph of G
   then let H' be the subgraph of G that is isomorphic to H and has the maximum weight over all subgraphs of G that are isomorphic to H, **return** H'
   else return $\emptyset$;
2. $\mathcal{P}_k = \emptyset$;
3. **loop** $2.51 \cdot 2^{rk} / \binom{k}{\lceil k/2 \rceil}$ times **do**
3.1. randomly partition the vertices of G into two parts V_L and V_R;
3.2. $\mathcal{P}_L = H$-**graph packing**$(G[V_L], \lceil k/2 \rceil)$; $\mathcal{P}_R = H$-**graph packing**$(G[V_R], \lfloor k/2 \rfloor)$;
3.3. **if** $\mathcal{P}_L \neq \emptyset$, $\mathcal{P}_R \neq \emptyset$, and the sum of the weights of $\mathcal{P}_L$ and $\mathcal{P}_R$ is larger than the weight of $\mathcal{P}_k$ **then** $\mathcal{P}_k = \mathcal{P}_L \cup \mathcal{P}_R$;
4. **return** $\mathcal{P}_k$.

FIG. 3. *A randomized divide-and-conquer algorithm for* P-MAX H-GRAPH PACKING.

in G and a vertex mapping from H to $G[V_r]$. Therefore, this process enumerates all possible isomorphic mappings from H to subgraphs of G. There are $\binom{n}{r}$ subsets of r vertices in G, and for each subset V_r, there are $r!$ vertex mappings from H to $G[V_r]$. For such a vertex mapping from H to $G[V_r]$, it takes time $O(r^2)$ to check if the mapping induces an isomorphic mapping from H to a subgraph of $G[V_r]$, assuming the graph G being given by an adjacency matrix. In summary, step 1 takes time $O(\binom{n}{r} r^2 r!) = O(n^r r^2)$. That is, we have

$$T(1, n) = O(n^r r^2).$$

The rest of the derivation of the running time of the algorithm follows directly from Corollary 2.3.

For space complexity, since the recursion depth is bounded by $O(\log k)$, and all copies of the graph H are disjoint, the space complexity of the algorithm is $O(rk \log k + n^2)$, where we assume that the graph G is given by an adjacency matrix that takes space $O(n^2)$. □

5. Derandomization. In this section, we discuss how the randomized algorithms presented in the previous sections can be derandomized. Our derandomization process is based on the construction of (n, k)-universal sets proposed by Naor, Schulman, and Srinivasan [17].

Assume that n and k are integers such that $n \geq k$. Denote by Z_n the set $\{0, 1, \ldots, n-1\}$. A *splitting function* over Z_n is a $\{0, 1\}$ (i.e., Boolean) function over Z_n. A subset S of Z_n is a *k-subset* if S consists of exactly k elements. Let (S_0, S_1) be a partition of the k-subset S, i.e., $S_0 \cup S_1 = S$ and $S_0 \cap S_1 = \emptyset$. We say that a splitting function f over Z_n *implements* the partition (S_0, S_1) if $f(x) = 0$ for all $x \in S_0$ and $f(y) = 1$ for all $y \in S_1$.

DEFINITION 5.1 (see [17]). *A set Ψ of splitting functions over Z_n is an (n, k)-*universal set *if for every k-subset S of Z_n and any partition (S_0, S_1) of S, there is a splitting function f in Ψ that implements (S_0, S_1). The* size *of an (n, k)-universal set Ψ is the number of splitting functions in Ψ.*

Naor, Schulman, and Srinivasan [17] developed a deterministic algorithm for the construction of an (n, k)-universal set. The construction was presented in an extended abstract [17] with many details omitted. Moreover, the construction was implicitly given via the construction of a more general structure, i.e., (n, k, l)-splitters. For the completeness of our discussion, we rephrase in the following proposition, more

precisely, a result on (n, k)-universal sets that is slightly different from the one given in [17]. Moreover, in the appendix, we present a detailed description for the construction of an (n, k)-universal set and give a formal proof of Proposition 5.2.[2]

PROPOSITION 5.2 (see [17]). *There is an $O(n2^{k+12\log^2 k})$ time deterministic algorithm that constructs an (n, k)-universal set of size bounded by $n2^{k+12\log^2 k+2}$.*

A function f on Z_n is *injective from a subset S of Z_n* if for any two different elements x and y in S, $f(x) \neq f(y)$.

By Bertrand's postulate, proved by Chebyshev in 1850 (see [22, section 5.2]), there is a prime number q such that $n \leq q < 2n$. Moreover, the smallest prime number q_0 between n and $2n$ can be constructed in time $O(n)$, as follows. By the prime number theorem (see [22, Theorem 5.19]), there is a constant d_0 such that for any integer $n \geq 2$, there is a prime number between n and $n + d_0 \log n$. Therefore, by checking each of the integers between n and $n + d_0 \log n$ and testing its primality using the trivial primality testing algorithm of running time $O(\sqrt{n})$, we can always find the smallest prime number q_0 between n and $2n$ in time $O(\sqrt{n} \log n) = O(n)$.

PROPOSITION 5.3 (see [8]). *Let n and k be integers, $n \geq k$, and let q_0 be the smallest prime number such that $n \leq q_0 < 2n$. For any k-subset S in Z_n, there is an integer z, $0 \leq z < q_0$, such that the function $g_{n,k,z}$ over Z_n, defined as $g_{n,k,z}(a) = (az \bmod q_0) \bmod k^2$, is injective from S.*

For a function $g_{n,k,z}$ from Z_n to $\{0, 1, \ldots, k^2 - 1\}$, as defined in Proposition 5.3, we say that the function $g_{n,k,z}$ *partitions* the set Z_n into k^2 pairwise disjoint subsets $\{W_0, W_1, \ldots, W_{k^2-1}\}$ if for all $0 \leq i \leq k^2 - 1$, $W_i = \{a \in Z_n \mid g_{n,k,z}(a) = i\}$.

Now we are ready for our derandomized algorithms. First consider the P-MAX PATH problem. Without loss of generality, we assume that the vertices in the input graph G are labeled by the integers $\{0, 1, \ldots, n - 1\}$. The algorithm for the P-MAX PATH problem is given in Figure 4.

THEOREM 5.4. *For a graph G of n vertices and m edges, the algorithm* **D-paths**(G, k) *solves the* P-MAX PATH *problem in time $4^{k+O(\log^3 k)}nm$.*

Proof. First consider the correctness of the algorithm. Note that the path ρ_0 returned by the algorithm **D-paths** remains $\emptyset$ unless step 4.2 of the algorithm replaces ρ_0 by a real k-path in G. In particular, the algorithm works correctly if the graph G contains no k-paths.

Now let ρ_k be a k-path of the maximum weight in the graph G. By Proposition 5.3, there is an integer z_0, $0 \leq z_0 < q_0$, such that the function g_{n,k,z_0} is injective from the k-subset consisting of the k vertices in the path ρ_k. Consider the execution of step 4.1 in algorithm **D-paths** on this particular integer z_0, which calls the subroutine **path-ext**$(\{\rho_\emptyset\}, V, z_0, k)$.

The subroutine **path-ext** has a structure similar to that of the algorithm **find-paths** in Figure 1, and our discussion will concentrate on the differences. For the integer z_0 and for any subset V' of V, we say that a path ρ is (z_0, V')-*separated* if for any two vertices u_1 and u_2 in ρ, where $u_1, u_2 \in V'$, we have $g_{n,k,z_0}(u_1) \neq g_{n,k,z_0}(u_2)$. For a general input (P_l, V', z_0, h) to the subroutine **path-ext**, we prove the following claim:

[2] As described in [17], it is possible to construct an (n, k)-universal set of size $2^{O(k)} \log n$. However, it is based on a more complicated construction that is also less efficient, compared to the one given in Proposition 5.2. Moreover, for most parameterized problems, it seems sufficient to use (n, k)-universal sets, where $n = O(k^2)$ (e.g., see the **D-paths** algorithm for the P-MAX PATH problem given in Figure 4). For this class of universal sets, the construction of Proposition 5.2 produces universal sets whose size is not worse than the one given in [17].

D-paths(G, k)
input: a weighted graph G with vertex set $V = \{0, 1, \ldots, n-1\}$, and an integer $k \geq 1$;
output: a k-path of the maximum weight in G if G contains k-paths;
1. **for** $h = 1$ to k **do** construct a (k^2, h)-universal set Ψ_h;
2. let q_0 be the smallest prime number such that $n \leq q_0 < 2n$;
3. $\rho_0 = \emptyset$; {suppose that $\emptyset$ is a virtual k-path of infinitely small weight}
4. **for** each z, $0 \leq z < q_0$ **do**
4.1. $P_k =$ **path-ext**$(\{\rho_\emptyset\}, V, z, k)$;
4.2. **if** P_k contains a k-path whose weight is larger than that of ρ_0
then replace ρ_0 by the k-path of the maximum weight in P_k;
5. **return** ρ_0.

path-ext(P_l, V', z, h)
input: a set P_l of l-paths in G; a subset V' of vertices in G (V' contains no vertex in P_l); an integer z that lets the function $g_{n,k,z}$ give an initial partition of V'; and an integer $h \geq 1$;
output: a set P_{l+h} of $(l+h)$-paths in $P_l \odot V'$;
1. $P_{l+h} = \emptyset$;
2. **if** $h = 1$ **then**
2.1. **if** $P_l = \{\rho_\emptyset\}$ **then** P_{l+1} contains a $(u, 1)$-path for each vertex $u \in V'$; **return** P_{l+1};
2.2. **else for** each (w, l)-path ρ_l in P_l and each $u \in V'$, where $[w, u]$ is an edge in G, **do**
2.3. concatenate ρ_l and u to make a $(u, l+1)$-path ρ_{l+1} in $P_l \odot V'$;
2.4. **if** P_{l+1} contains no $(u, l+1)$-path **then** add ρ_{l+1} to P_{l+1};
2.5. **else if** the $(u, l+1)$-path ρ'_{l+1} in P_{l+1} has a weight smaller than that of ρ_{l+1}
2.6. **then** replace ρ'_{l+1} in P_{l+1} by ρ_{l+1};
2.7. **return** P_{l+1};
3. **for** each splitting function f in the (k^2, h)-universal set Ψ_h **do**
3.1. $V_L = \{v, \mid v \in V'$ and $f(g_{n,k,z}(v)) = 0\}$;
3.2. $V_R = \{v, \mid v \in V'$ and $f(g_{n,k,z}(v)) = 1\}$;
3.3. $P^L_{l+\lceil h/2 \rceil} =$ **path-ext**$(P_l, V_L, z, \lceil h/2 \rceil)$;
3.4. **if** $P^L_{l+\lceil h/2 \rceil} \neq \emptyset$ **then**
3.5. $P^R_{l+h} =$ **path-ext**$(P^L_{l+\lceil h/2 \rceil}, V_R, z, \lfloor h/2 \rfloor)$;
3.6. **for** each $(u, l+h)$-path ρ_{l+h} in P^R_{l+h} **do**
3.7. **if** P_{l+h} contains no $(u, l+h)$-path in $P_l \odot V'$ **then** add ρ_{l+h} to P_{l+h};
3.8. **else if** the $(u, l+h)$-path ρ'_{l+h} in P_{l+h} has a weight smaller than that of ρ_{l+h}
3.9. **then** replace ρ'_{l+h} in P_{l+h} by ρ_{l+h};
4. **return** P_{l+h}.

FIG. 4. *A deterministic algorithm for* P-MAX PATH.

> For any vertex $v \in V'$, there is a (z_0, V')-separated $(v, l+h)$-path in $P_l \odot V'$ if and only if the set P_{l+h} constructed by the subroutine **path-ext**(P_l, V', z_0, h) contains a (z_0, V')-separated $(v, l+h)$-path in $P_l \odot V'$ whose weight is the maximum over all (z_0, V')-separated $(v, l+h)$-paths in $P_l \odot V'$.

One direction is trivial: it suffices to ensure the existence of a (z_0, V')-separated $(v, l+h)$-path in $P_l \odot V'$ if the set P_{l+h} contains a (z_0, V')-separated $(v, l+h)$-path in $P_l \odot V'$. We prove the other direction by induction on h. For the case $h = 1$, the algorithm **path-ext** proceeds in exactly the same way as that of the algorithm **find-paths** in Figure 1. Since every $(v, l+1)$-path in $P_l \odot V'$ is (z_0, V')-separated, in this case the corresponding part of the proof for algorithm **find-paths** (i.e., Theorem 3.1) is directly applied to derive the correctness of the above claim. Now consider the case $h > 1$. Suppose that there is a (z_0, V')-separated $(v, l+h)$-path in $P_l \odot V'$, and let

$$\rho_{l+h} = \langle u_1, \ldots, u_l, w_1, \ldots, w_{h_1}, \ldots, w_h \rangle$$

be a (z_0, V')-separated $(v, l+h)$-path of the maximum weight in $P_l \odot V'$, where $\langle u_1, \ldots, u_l \rangle$ is a path in P_l, $w_j \in V'$ for all j, $h_1 = \lceil h/2 \rceil$, and $w_h = v$. Since ρ_{l+h} is (z_0, V')-separated, $S_h = \{g_{n,k,z_0}(w_1), \ldots, g_{n,k,z_0}(w_h)\}$ consists of exactly h elements

in the set Z_{k^2}, and $(\{g_{n,k,z_0}(w_1),\ldots,g_{n,k,z_0}(w_{h_1})\},\{g_{n,k,z_0}(w_{h_1+1}),\ldots,g_{n,k,z_0}(w_h)\})$ is a partition of S_h. By the definition of the (k^2,h)-universal set Ψ_h, there is a splitting function f_0 in Ψ_h such that $f_0(g_{n,k,z_0}(w_j))=0$ for $1\le j\le h_1$, and $f_0(g_{n,k,z_0}(w_j))=1$ for $h_1+1\le j\le h$. Therefore, when this splitting function f_0 is picked in step 3 of the algorithm **path-ext**(P_l,V',z_0,h), the set V_L obtained in step 3.1 contains the vertices $w_1,\ldots,w_{h_1}$, and the set V_R obtained in step 3.2 contains the vertices $w_{h_1+1},\ldots,w_h$.

Note that the path $\rho_{l+h_1}=\langle u_1,\ldots,u_l,w_1,\ldots,w_{h_1}\rangle$ is a (z_0,V_L)-separated $(w_{h_1},l+h_1)$-path in $P_l\odot V_L$. By the induction hypothesis, the set $P^L_{l+h_1}$ obtained in step 3.3 contains a (z_0,V_L)-separated $(w_{h_1},l+h_1)$-path $\overline{\rho}_{l+h_1}$ in $P_l\odot V_L$ whose weight is at least as large as that of ρ_{l+h_1}. Now the concatenation of the path $\overline{\rho}_{l+h_1}$ and the path $\langle w_{h_1+1},\ldots,w_h\rangle$ is a (z_0,V_R)-separated $(w_h,(l+h_1)+(h-h_1))$-path (i.e., a $(v,l+h)$-path) in $P^L_{l+h_1}\odot V_R$. Thus, by our induction hypothesis again, the set P^R_{l+h} obtained in step 3.5 contains a (z_0,V_R)-separated $(v,l+h)$-path $\overline{\rho}_{l+h}$ in $P^L_{l+h_1}\odot V_R$ whose weight is at least as large as the sum of the weights of the paths $\overline{\rho}_{l+h_1}$ and $\langle w_{h_1+1},\ldots,w_h\rangle$. Since the weight of $\overline{\rho}_{l+h_1}$ is not smaller than that of ρ_{l+h_1}, we conclude that the weight of the path $\overline{\rho}_{l+h}$ is not smaller than that of ρ_{l+h}. Finally, since the path $\overline{\rho}_{l+h}$ is a concatenation of a $(w,l+h_1)$-path ρ'_{l+h_1} in $P^L_{l+h_1}$ and a path in $G[V_R]$ that is (z_0,V_R)-separated, where by the induction hypothesis, ρ'_{l+h_1} is a (z_0,V_L)-separated path in $P_l\odot V_L$, we derive that $\overline{\rho}_{l+h}$ is actually a (z_0,V')-separated $(v,l+h)$-path in $P_l\odot V'$ (note that for two vertices $w\in V_L$ and $w'\in V_R$, the values $g_{n,k,z_0}(w)$ and $g_{n,k,z_0}(w')$ are always different because they are mapped to different values under the splitting function f_0). Since the weight of $\overline{\rho}_{l+h}$ is not smaller than the weight of ρ_{l+h}, and by our assumption, the path ρ_{l+h} has the maximum weight over all (z_0,V')-separated $(v,l+h)$-paths in $P_l\odot V'$, we conclude that $\overline{\rho}_{l+h}$ must also be a (z_0,V')-separated $(v,l+h)$-path of the maximum weight in $P_l\odot V'$. Therefore, the collection P_{l+h} returned by the subroutine **path-ext**(P_l,V',z_0,h) must contain a (z_0,V')-separated $(v,l+h)$-path of the maximum weight. This completes the proof for the claim.

Now the correctness of the algorithm **D-paths** can be easily derived from the above claim: by our assumption on the integer z_0, the k-path ρ_k of the maximum weight in the graph G is actually a (z_0,V)-separated (v,k)-path in $\{\rho_\emptyset\}\odot V$ (i.e., in the graph G). Therefore, the above claim concludes that the set P_k obtained in step 4.1 of the algorithm **D-paths** by calling the subroutine **path-ext**$(\{\rho_\emptyset\},V,z_0,k)$ must contain a (v,k)-path of the maximum weight. In consequence, the path ρ_0 returned in step 5 of the algorithm **D-paths**(G,k) must be a k-path of the maximum weight.

For the running time of the algorithm, note that the running time of the algorithm **D-paths** is dominated by step 4, which is a q_0-time iteration of the subroutine **path-ext**, where $q_0=O(n)$. Let $T(h,m)$ be the running time of the recursive subroutine **path-ext**(P_l,V',z,h), where m is the number of edges in the input graph G to the main algorithm **D-paths**(G,k). By Proposition 5.2, the (k^2,h)-universal set Ψ_h has at most $k^2 2^{h+12\log^2 h+2}$ splitting functions. Therefore, the value $T(h,m)$ satisfies the following recurrence relations (where c_1 and c_2 are constants):

$$\begin{aligned} &T(1,m)\le c_1 m;\\ &T(h,m)\le k^2 2^{h+12\log^2 h+2}[c_2 m+T(\lceil h/2\rceil,m)+T(\lfloor h/2\rfloor,m)]. \end{aligned}\tag{3}$$

Let $X_0=k^2 2^{12\log^2 k+2}=2^{12\log^2 k+2\log k+2}$. Since $k\ge h$ for all values h used in the subroutine **path-ext**(P_l,V',z,h), we derive from (3) that

$$\begin{aligned} &T(1,m)\le c_1 m;\\ &T(h,m)\le 2^h X_0[c_2 m+T(\lceil h/2\rceil,m)+T(\lfloor h/2\rfloor,m)]. \end{aligned}\tag{4}$$

Since the value X_0 is independent of the variables h and m, we can apply Corollary 2.2 to the recurrence relations in (4) and obtain $T(h,m) = O(4^h h^\alpha m)$, where α is any number larger than $\log_2(X_0(4+1)/2)$. In particular, if we pick $\alpha = \log_2 X_0 + 2 = 12\log^2 k + 2\log k + 4$, we have

$$h^\alpha \leq k^\alpha = 2^{\alpha \log k} = 2^{O(\log^3 k)},$$

which gives

$$T(h,m) = O(4^h 2^{O(\log^3 k)} m) = 4^{k+O(\log^3 k)} m.$$

Combining this with our previous discussion, we conclude that the running time of the algorithm **D-paths** is bounded by $4^{k+O(\log^3 k)} nm$. □

Using the same technique, we can develop improved deterministic algorithms for the matching and packing problems discussed in section 4.

THEOREM 5.5. *There is a deterministic algorithm that solves the* P-MAX r-SET PACKING *problem in time* $4^{rk+O(\log^3(rk))} n^2$, *where* n *is the number of* r*-sets in the input instance.*

Proof. The deterministic algorithm for the P-MAX r-SET PACKING problem is a derandomization of the algorithm given in Figure 2. The procedure is very similar to that described in Theorem 5.4 for the P-MAX PATH problem. The only difference is that here we use an $((rk)^2, rh)$-universal set in which a splitting function splits the (implicit) h-packing of the maximum weight into two packings of sizes $\lceil h/2 \rceil$ and $\lfloor h/2 \rfloor$, respectively. We leave the details for interested readers to verify. □

COROLLARY 5.6. *There is a deterministic algorithm that solves the* P-MAX r-D MATCHING *problem in time* $4^{rk+O(\log^3(rk))} n^2$, *where* n *is the number of points in the input instance.*

COROLLARY 5.7. *Let* H *be a fixed graph of* r *vertices. There is a deterministic algorithm that solves the* P-MAX H-GRAPH PACKING *problem in time* $4^{rk+O(\log^3(rk))} n^{r+1}$, *where* n *is the number of vertices in the input graph* G.

Remark 1. The results in this section significantly improve previous best deterministic algorithms for the problems [2, 7, 13]. In fact, the running time of our algorithms is even better than that of the previous best randomized algorithms for the corresponding problems. The previous best randomized algorithm for the P-MAX PATH problem has running time $O(5.44^k km)$ [2], and previous best randomized algorithms for the P-MAX r-SET PACKING and P-MAX r-D MATCHING problems have running time $10.88^{rk} n^{O(1)}$ [13].

Remark 2. We noted that very recently Koutis [14] reported a randomized algorithm of running time $O(2^{3k/2} n^{O(1)})$ for the P-MAX PATH problem, and randomized algorithms of running time $O(2^{3k} n^{O(1)})$ for the P-MAX 3-SET PACKING and P-MAX 3-D MATCHING problems. The running time of Koutis's randomized algorithm for the P-MAX PATH problem has even been further improved by Williams to $O(2^k n^{O(1)})$ [23]. However, as remarked in [23], the algorithms reported in [14, 23] work only for the unweighted versions of the problems and do not appear to extend to the more general weighted versions. Moreover, it is unknown, as indicated in [14], whether the new randomized algorithms for P-MAX 3-SET PACKING and P-MAX 3-D MATCHING can be extended to P-MAX r-SET PACKING and P-MAX r-D MATCHING for general $r > 3$. Finally, as asked in both [14] and [23], it is not clear whether the new randomized algorithms can be derandomized to result in improvements over the deterministic algorithms presented in the current paper.

Appendix. Deterministic construction of (n,k)-universal sets. Naor, Schulman, and Srinivasan developed a deterministic construction of (n,k)-universal sets [17]. The construction was described via the construction of a more general structure, i.e., (n,k,l)-splitters. Moreover, the construction was presented in an extended abstract [17] in which many details were omitted. For the completeness of our discussion, we will reproduce in this appendix a slightly different and simpler construction and its analysis specifically for (n,k)-universal sets. The presentation here is more precise with all needed details provided.

We start with some terminology and definitions in probability theory.

Let $(\Omega, \Pr)$ be a probability space, where Ω is a finite set and $\Pr$ is the probability measure. The *size* of $(\Omega, \Pr)$ is the number of elements in Ω. The probability space $(\Omega, \Pr)$ is *uniform* if $\Pr(a) = 1/|\Omega|$ for all $a \in \Omega$ (in this case, we will simply write the probability space as Ω).

A $\{0,1\}$-*random variable* ξ over the probability space $(\Omega, \Pr)$ is a function from Ω to $\{0,1\}$. A group of h $\{0,1\}$-random variables $\xi_1, \xi_2, \ldots, \xi_h$ are *mutually independent* if for any combination of h binary bits $b_1, \ldots, b_h$ in $\{0,1\}$ the following holds:

$$\Pr(\xi_1 = b_1, \xi_2 = b_2, \ldots, \xi_h = b_h) = \Pr(\xi_1 = b_1)\Pr(\xi_2 = b_2) \cdots \Pr(\xi_h = b_h).$$

A group of n $\{0,1\}$-random variables $\xi_1, \xi_2, \ldots, \xi_n$ are *k-wise independent* if every group of k different $\{0,1\}$-random variables among $\xi_1, \xi_2, \ldots, \xi_n$ are mutually independent.

The following lemma is crucial to our construction and was first proved in [1].

LEMMA A.1 (see [1]). *Let $n = 2^d - 1$ for an integer d and let k be an odd number, $k \le n$. There is an algorithm of running time $O(n(n+1)^{(k-1)/2})$ that constructs a uniform probability space Ω of size $2(n+1)^{(k-1)/2}$ and a group of n k-wise independent $\{0,1\}$-random variables $\xi_1, \ldots, \xi_n$ over Ω such that $Pr(\xi_i = 0) = Pr(\xi_i = 1) = 1/2$ for all $1 \le i \le n$.*

We start with a simple observation.

LEMMA A.2. *For every integer $n \ge 1$, there is an $(n,1)$-universal set of size 2.*

Proof. The splitting functions $f_0 \equiv 0$ and $f_1 \equiv 1$ over Z_n obviously form an $(n,1)$-universal set for Z_n. ∎

Now we present a construction of a general (n,k)-universal set of small size that, however, is not sufficiently efficient. Compared to the work presented in [17], the size of our structure is more precise (and slightly improved), which will be important for the later construction. Moreover, the time complexity of our construction is lower than that described in [17].

LEMMA A.3. *Let k be an odd number and let n be an integer such that $n \ge k$ and $n \ge 2$. There is an (n,k)-universal set of size bounded by $2^k k \log n$, which can be constructed in time $O\left(\binom{n}{k} 2^k k (2n)^{(k-1)/2}\right)$.*

Proof. The lemma is true for $k = 1$ by Lemma A.2. Thus, we assume $k \ge 3$.

Let $n_1 = 2^d - 1$, where d is the smallest integer such that $n \le n_1$ (note that $n \le n_1 \le 2n-1$). By Lemma A.1, we can construct, in time $O(n_1(n_1+1)^{(k-1)/2})$, a uniform probability space Ω of size $2(n_1+1)^{(k-1)/2}$ and a group of n_1 k-wise independent $\{0,1\}$-random variables $\xi_1, \ldots, \xi_{n_1}$ over Ω such that $\Pr(\xi_i = 0) = \Pr(\xi_i = 1) = 1/2$ for all $1 \le i \le n_1$. By picking the first n of these n_1 random variables, we get a group of n k-wise independent $\{0,1\}$-random variables $\xi_1, \ldots, \xi_n$ over the uniform probability space Ω such that $\Pr(\xi_i = 0) = \Pr(\xi_i = 1) = 1/2$ for all $1 \le i \le n$. All these can be constructed in time $O(n(2n)^{(k-1)/2})$.

Note that the uniform probability space Ω and the random variables $\xi_1, \ldots, \xi_n$ constructed above actually make a collection $\mathcal{P}$ of $D = 2(n_1+1)^{(k-1)/2}$ splitting

functions over Z_n. In fact, for each element a in Ω, the values of the random variables $\xi_1, \ldots, \xi_n$ make a binary string $\xi_1(a) \cdots \xi_n(a)$ of length n, which can be interpreted as a splitting function over Z_n.

Construct a bipartite graph $G = (V_1 \cup V_2, E)$ with the vertex bipartition (V_1, V_2), where V_1 consists of D vertices, corresponding to the D splitting functions in the collection $\mathcal{P}$, and the vertex set V_2 consists of $D' = \binom{n}{k}2^k$ vertices such that for each k-subset S of Z_n and each partition (S_1, S_2) of S, there is a corresponding vertex in V_2. An edge $[v, w]$ is created in G if the splitting function corresponding to the vertex $v \in V_1$ implements the partition (S_1, S_2) of a k-subset S of Z_n that correspond to the vertex $w \in V_2$.

CLAIM. *Each vertex in V_2 has a degree $D/2^k$.*

Proof of the claim. Let $S = \{h_1, \ldots, h_k\}$ be any k-subset of Z_n and let (S_1, S_2) be a partition of S. Define k binary bits b_{h_i}, $1 \le i \le k$, such that $b_{h_i} = 0$ if $h_i \in S_1$ and $b_{h_i} = 1$ if $h_i \in S_2$. Consider the k mutually independent random variables $\xi_{h_1}, \ldots, \xi_{h_k}$ (they are mutually independent because the random variables $\xi_1, \ldots, \xi_n$ are k-wise independent); we have

$$\Pr(\xi_{h_1} = b_{h_1}, \ldots, \xi_{h_k} = b_{h_k}) = \Pr(\xi_{h_1} = b_{h_1}) \cdots \Pr(\xi_{h_k} = b_{h_k}) = 1/2^k.$$

Thus, there are $D/2^k$ elements a in Ω such that $\xi_{h_i}(a) = b_{h_i}$ for $1 \le i \le k$. By the interpretation above, there are $D/2^k$ splitting functions in the collection $\mathcal{P}$ that implement the partition (S_1, S_2) of the k-subset S. By our construction of the graph G, the vertex w in V_2 corresponding to the partition (S_1, S_2) of the k-subset S has degree $D/2^k$. The claim now is proved because S is an arbitrary k-subset in Z_n and (S_1, S_2) is an arbitrary partition of S. □

Since there are D' vertices in V_2, the above claim shows that the bipartite graph G contains exactly $(D'D)/2^k$ edges. Now since there are D vertices in V_1, there is a vertex v_1 in V_1 in the graph G whose degree is at least $D'/2^k$. In other words, there is a splitting function in the collection $\mathcal{P}$ that implements at least $D'/2^k$ partitions of k-subsets of Z_n (these partitions can be partitions for different k-subsets of Z_n).

We perform the following operations on the graph G: mark the vertex v_1 in V_1 and remove all vertices in V_2 that are adjacent to v_1 (or, equivalently, we mark a splitting function f in $\mathcal{P}$ and remove all partitions of k-subsets of Z_n that are implemented by f). Let D'_1 be the number of vertices in V_2 in the remaining bipartite graph G'. $D'_1 \le (1 - 1/2^k)D'$.

Note that each vertex in V_2 in the remaining graph G' still has degree $D/2^k$. Therefore, by repeating the above process, in the remaining graph G' we can find a vertex v_2 in V_1 that is adjacent to at least $D'_1/2^k$ vertices in V_2. Now we mark v_2 and remove the vertices in V_2 that are adjacent to v_2. Now there are at most $D'_2 \le (1 - 1/2^k)D'_1 \le (1 - 1/2^k)^2 D'$ vertices in V_2 in the remaining graph.

Repeat the above process until all vertices in V_2 are removed. The number t' of times the above process is repeated is not larger than the smallest integer t such that $(1 - 1/2^k)^t D' < 1$. For $k = 3$, we can directly verify that $t' \le 2^k k \log n$; and for $k \ge 5$, since $D' = \binom{n}{k}2^k \le n^k$ and $(1 - 1/2^k)^{2^k} < 1/e$, we can also formally prove that $t' \le 2^k k \log n$.

Each execution of the above process marks a vertex in V_1; therefore, there are at most $2^k k \log n$ vertices in V_1 that are marked in the above process. By our construction, all vertices in the set V_2 are adjacent to at least one marked vertex in V_1. Accordingly, there are $D'' \le 2^k k \log n$ splitting functions in the collection $\mathcal{P}$ such that every partition of any k-subset in Z_n is implemented by at least one of these D''

splitting functions. That is, these D'' splitting functions make an (n,k)-universal set $\mathcal{P}'$ of size bounded by $2^k k \log n$.

We analyze the time complexity for the entire construction of the (n,k)-universal set $\mathcal{P}'$. As given earlier, the construction of the uniform probability space Ω and the $\{0,1\}$-random variables $\xi_1,\ldots,\xi_n$ takes time $O(n(2n)^{(k-1)/2})$. The construction of the bipartite graph G takes time $O(k|V_1||V_2|) = O(kDD') = O\big(\binom{n}{k}k2^k(2n)^{(k-1)/2}\big)$. To perform the above iteration process of marking vertices in V_1, we can represent the bipartite graph G as a $D \times D'$ matrix and keep an array for the degrees of the vertices in V_1. It is not difficult to verify that on such data structures, the entire vertex marking process takes time $O(t'(D+D') + DD') = O\big(\binom{n}{k}2^k(2n)^{(k-1)/2}\big)$. Thus, the total construction of the (n,k)-universal set $\mathcal{P}'$ takes time $O\big(\binom{n}{k}2^k k(2n)^{(k-1)/2}\big)$. □

The size of the (n,k)-universal set constructed in Lemma A.3 is quite small. Unfortunately, the time complexity for constructing such an (n,k)-universal set given in the lemma is unacceptably high. Therefore, we need to use additional techniques to reduce the construction time.

Fix n and k, where $n \geq k$. At the moment, we assume $k \geq 2$. Define

$$
\begin{aligned}
&k_1 = \text{ the largest odd number bounded by } k/(4\log k),\\
&t = \lceil k/k_1 \rceil \qquad \text{(it is not hard to verify that } t \leq 4\log k + 2),\\
&k_2 = k - k_1(t-1) \qquad \text{(note that } k_2 \leq k_1),\\
&n_1 = k^2, \text{ and}\\
&p = \text{ a prime number such that } n \leq p < 2n,
\end{aligned}
\tag{5}
$$

where the existence of the prime number p above is guaranteed by Bertrand's postulate [22].

Consider the set $Z_{k^2} = \{0,1,\ldots,k^2-1\}$. Pick any $t-1$ elements $i_2, i_3, \ldots, i_t$ in Z_{k^2}, such that $i_2 < i_3 < \cdots < i_t$. These $t-1$ elements naturally divide the set Z_{k^2} into t sets consisting of consecutive elements (where X_1 may be an empty set):

$$X_1 = \{0,\ldots,i_2-1\}, \quad X_2 = \{i_2,\ldots,i_3-1\}, \quad \ldots, \quad X_t = \{i_t,\ldots,k^2-1\}.$$

Such a division $(X_1, X_2, \ldots, X_t)$ of the set Z_{k^2} based on $t-1$ selected elements in Z_{k^2} will be called a *t-grouping* of the set Z_{k^2}.

According to Lemma A.3, we can construct (note that k_1 is an odd number and that $n_1 \geq 4$) an (n_1,k_1)-universal set $\mathcal{P}_1$ of size $D_1 \leq k_1 2^{k_1}\log n_1$ in time $O(\binom{n_1}{k_1}2^{k_1}k_1(2n_1)^{(k_1-1)/2})$. Moreover, we define $k_2' = k_2$ if k_2 is odd, and $k_2' = k_2+1$ if k_2 is even, and we construct an (n_1,k_2')-universal set $\mathcal{P}_2$ of size $D_2 \leq k_2' 2^{k_2'}\log n_1 \leq k_1 2^{k_2+1}\log n_1$ in time $O(\binom{n_1}{k_1}2^{k_1}k_1(2n_1)^{(k_1-1)/2})$ (we can replace k_2' by k_1 because by definition $k_2' \leq k_1$). Using the definitions of k_1, k_2, and n_1, it is not hard to verify that

$$D_1 \leq k2^{k_1-1} \qquad \text{and} \qquad D_2 \leq k2^{k_2}. \tag{6}$$

LEMMA A.4. *Let $\mathcal{P}$ be an (n,k)-universal set. Then for any n', $k \leq n' \leq n$, $\mathcal{P}$ is also an (n',k)-universal set; and for any $k' \leq k$, $\mathcal{P}$ is also an (n,k')-universal set.*

Proof. Each splitting function in $\mathcal{P}$ can be regarded as a splitting function over $Z_{n'}$. Since every k-subset of $Z_{n'}$ is also a k-subset of Z_n, we conclude that any partition of any k-subset in $Z_{n'}$ is implemented by a splitting function in $\mathcal{P}$; i.e., $\mathcal{P}$ is also an (n',k)-universal set.

Splitting $f_{z,(X_1,\ldots,X_t),(f_1,\ldots,f_{t-1},f_t)}(a)$
where $a \in Z_n$ is the input of the function, $0 \le z < p$, $(X_1, \ldots, X_t)$ is a t-grouping of Z_{k^2}, $f_1, \ldots, f_{t-1}$ are splitting functions in $\mathcal{P}_1$, and f_t is a splitting function in $\mathcal{P}_2$.
1. $x = (az \bmod p) \bmod k^2$;
2. suppose that x is the jth smallest element in X_i;
3. return $f_i(j)$.

FIG. 5. *A splitting function over* Z_n.

Every partition (S_1', S_2') of any k'-subset S' of Z_n can be extended to a partition (S_1, S_2) of a k-subset of Z_n by adding $k-k'$ elements in $Z_n - S'$ to S_1'. Now the splitting function in $\mathcal{P}$ that implements (S_1, S_2) also implements the partition (S_1', S_2') of S'. Thus, $\mathcal{P}$ is also an (n, k')-universal set. □

Now we are ready to construct our (n, k)-universal set $\mathcal{P}$. Each splitting function over Z_n in $\mathcal{P}$ is defined based on an integer z between 0 and $p-1$, a t-grouping $(X_1, \ldots, X_t)$ of the set Z_{k^2}, $t-1$ splitting functions $f_1, \ldots, f_{t-1}$ in the (n_1, k_1)-universal set $\mathcal{P}_1$, and a splitting function f_t in the (n_1, k_2')-universal set $\mathcal{P}_2$. The splitting function over Z_n is defined by the algorithm given in Figure 5.

First note that the function $f_{z,(X_1,\ldots,X_t),(f_1,\ldots,f_{t-1},f_t)}(a)$ is a well-defined splitting function. In fact, by step 1, x is an element in Z_{k^2}. Since $(X_1, \ldots, X_t)$ is a t-grouping of Z_{k^2}, x must belong to a unique X_i and have a unique rank j in X_i. Thus, step 3 will return a Boolean value $f_i(j)$.

DEFINITION A.5. *Let $\mathcal{P}$ be the collection of all possible splitting functions over Z_n defined in Figure 5, over all integers z, $0 \le z < p$, all t-groupings $(X_1, \ldots, X_t)$ of Z_{k^2}, all possible lists $(f_1, \ldots, f_{t-1})$ of splitting functions in the (n_1, k_1)-universal set $\mathcal{P}_1$ (where the same function may appear more than once in the list), and all splitting functions f_t in the (n_1, k_2')-universal set $\mathcal{P}_2$.*

Now we are ready for our main result.

THEOREM A.6 (see [17]). *For all integers $k \ge 1$ and $n \ge k$, there is an (n, k)-universal set of size bounded by $n2^{k+12\log^2 k+2}$, which can be constructed in time $O(n2^{k+12\log^2 k})$.*

Proof. The theorem holds true for $k = 1$ by Lemma A.2. Thus, we assume $k \ge 2$. We prove that the collection $\mathcal{P}$ given in the definition before this theorem is an (n, k)-universal set that satisfies the conditions given in the theorem.

We first consider the size of the collection $\mathcal{P}$. There are $p < 2n$ possible integers z. As described earlier, each t-grouping of Z_{k^2} can be given by $t-1$ different elements in Z_{k^2}. Therefore, the total number of different t-groupings of Z_{k^2} is bounded by $\binom{k^2}{t-1} \le k^{2(t-1)}$. The number of possible lists $(f_1, \ldots, f_{t-1})$ of splitting functions in $\mathcal{P}_1$ is $|\mathcal{P}_1|^{t-1} = D_1^{t-1}$, and, finally, the number of splitting functions in $\mathcal{P}_2$ is $|\mathcal{P}_2| = D_2$. Putting all these together, and recalling the definitions and inequalities in (5) and (6), we conclude that the size of $\mathcal{P}$ is bounded by

$$\begin{aligned}
&2nk^{2(t-1)} D_1^{t-1} D_2 \\
&\le 2nk^{2(t-1)} (k2^{k_1-1})^{t-1} (k2^{k_2}) \\
&= 2nk^{3t-2} 2^{(k_1-1)(t-1)+k_2} \\
&\le 2nk^{12\log k+4} 2^{(k_1-1)(t-1)+(k-k_1(t-1))} \\
&= n2^{12\log^2 k+4\log k+1} 2^{k-(t-1)} \\
&\le n2^{k+12\log^2 k+2},
\end{aligned}$$

where the last inequality has used the facts $t = \lceil k/k_1 \rceil \ge k/k_1 \ge k/(k/(4\log k)) = 4\log k$.

To construct the collection $\mathcal{P}$, we first construct the collections $\mathcal{P}_1$ and $\mathcal{P}_2$. As discussed earlier, $\mathcal{P}_1$ and $\mathcal{P}_2$ can be constructed in time $O(\binom{n_1}{k_1}2^{k_1}k_1(2n_1)^{(k_1-1)/2}) = O(2^k)$. Once the collections $\mathcal{P}_1$ and $\mathcal{P}_2$ are available, the integers z, the t-groupings $(X_1, \ldots, X_t)$ of Z_{k^2}, and the lists $(f_1, \ldots, f_{t-1})$ of splitting functions in $\mathcal{P}_1$ and the splitting functions f_t in $\mathcal{P}_2$ can be systematically enumerated, in constant time per combination, which gives a representation of the corresponding splitting function in $\mathcal{P}$. In conclusion, the collection $\mathcal{P}$ can be constructed in time $O(|\mathcal{P}|) = O(n2^{k+12\log^2 k})$.

What remains is to show that $\mathcal{P}$ is an (n,k)-universal set. For this, let S be a given k-subset of Z_n and let (S_1, S_2) be a partition of S. By Proposition 5.3, there is an integer z_0, $0 \le z_0 < p$, such that the function g_{n,k,z_0} over Z_n is injective from S. Let S', S'_1, and S'_2 be the subsets of Z_{k^2} that are the images of S, S_1, and S_2 under g_{n,k,z_0}, respectively. By the definitions, we have $|S'| = |S|$, $|S'_1| = |S_1|$, and $|S'_2| = |S_2|$, and (S'_1, S'_2) is a partition of the k-subset S' in Z_{k^2}.

It is easy to see that there is a t-grouping $(X^0_1, \ldots, X^0_t)$ of the set Z_{k^2} such that each of the first $t-1$ subsets $X^0_1, \ldots, X^0_{t-1}$ contains exactly k_1 elements in S', and the last subset X^0_t contains k_2 elements in S'. Let $T_i = X^0_i \cap S'$ for $1 \le i \le t$. Then T_i is a k_1-subset of X^0_i for $1 \le i \le t-1$, and T_t is a k_2-subset of X^0_t. Moreover, the partition (S'_1, S'_2) of S' induces a partition $(T_{i,1}, T_{i,2})$ for each T_i, $1 \le i \le t$, where $T_{i,1} = T_i \cap S'_1$ and $T_{i,2} = T_i \cap S'_2$.

Since $\mathcal{P}_1$ is an (n_1, k_1)-universal set, which by Lemma A.4 is also an $(|X^0_i|, k_1)$-universal set, for each i, $1 \le i \le t-1$, there is a splitting function f^0_i in $\mathcal{P}_1$ that implements the partition $(T_{i,1}, T_{i,2})$ of the k_1-subset T_i of X^0_i (note that the subset X^0_i can be regarded as the set $Z_{|X^0_i|}$) for $1 \le i \le t-1$. That is, $f^0_i(x) = 0$ if $x \in T_{i,1}$ and $f^0_i(y) = 1$ if $y \in T_{i,2}$. Similarly, there is a splitting function f^0_t in $\mathcal{P}_2$ that implements the partition $(T_{t,1}, T_{t,2})$ of the k_2-subset T_t.

Now consider the splitting function $f_{z_0,(X^0_1,\ldots,X^0_t),(f^0_1,\ldots,f^0_t)}$. On an element a in the subset S_1, step 1 of the algorithm **Splitting** produces an element $x = g_{n,k,z_0}(a)$ in the set S'_1. Suppose that x is in the set X^0_i; then x is in the set $T_{i,1}$. By the way we selected the splitting function f^0_i, we have $f^0_i(x) = 0$. In summary, on an element a in the subset S_1, we have $f_{z_0,(X^0_1,\ldots,X^0_t),(f^0_1,\ldots,f^0_t)}(a) = 0$. Using the same reasoning, we can show $f_{z_0,(X^0_1,\ldots,X^0_t),(f^0_1,\ldots,f^0_t)}(a) = 1$ for every element a in S_2. Therefore, the function $f_{z_0,(X^0_1,\ldots,X^0_t),(f^0_1,\ldots,f^0_t)}$ in the collection $\mathcal{P}$ implements the partition (S_1, S_2) of the k-subset S of Z_n.

Since S is an arbitrary k-subset of Z_n and (S_1, S_2) is an arbitrary partition of S, we conclude that the collection $\mathcal{P}$ is an (n,k)-universal set. □

Acknowledgments. The authors are grateful to Mike Fellows, Fedor Fomin, Somit Gupta, Ming Li, and Vijaya Ramachandran for their comments and discussions on early versions of this paper. The authors would also like to thank the referees for their critical reading, helpful comments, and constructive suggestions, which have resulted in improvements on the presentations and certain technical results.

REFERENCES

[1] N. Alon, L. Babai, and A. Itai, *A fast and simple randomized parallel algorithm for the maximal independent set problem*, J. Algorithms, 7 (1986), pp. 567–683.

[2] N. Alon, R. Yuster, and U. Zwick, *Color-coding*, J. ACM, 42 (1995), pp. 844–856.

[3] H. Bodlaender, *On linear time minor tests with depth-first search*, J. Algorithms, 14 (1993), pp. 1–23.

[4] J. Chen, D. Friesen, W. Jia, and I. Kanj, *Using nondeterminism to design efficient deterministic algorithms*, Algorithmica, 40 (2004), pp. 83–97.

[5] J. Chen, S. Lu, S.-H. Sze, and F. Zhang, *Improved algorithms for path, matching, and packing problems*, in Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2007), SIAM, Philadelphia, 2007, pp. 298–307.

[6] R. Downey and M. Fellows, *Parameterized Complexity*, Springer, New York, 1999.

[7] M. Fellows, C. Knauer, N. Nishimura, P. Ragde, F. Rosamond, U. Stege, D. Thilikos, and S. Whitesides, *Faster fixed-parameter tractable algorithms for matching and packing problems*, in Algorithms–ESA 2004, Lecture Notes in Comput. Sci. 3221, Springer, Berlin, 2004, pp. 311–322.

[8] M. Fredman, J. Komlos, and E. Szemeredi, *Storing a sparse table with $O(1)$ worst case access time*, J. ACM, 31 (1984), pp. 538–544.

[9] R. Graham, D. Knuth, and O. Patashnik, *Concrete Mathematics*, 2nd ed., Addison–Wesley, Reading, MA, 1994.

[10] W. Jia, C. Zhang, and J. Chen, *An efficient parameterized algorithm for m-set packing*, J. Algorithms, 50 (2004), pp. 106–117.

[11] B. Kelley, R. Sharan, R. Karp, T. Sittler, D. Root, B. Stockwell, and T. Ideker, *Conserved pathways within bacteria and yeast as revealed by global protein network alignment*, Proc. Natl. Acad. Sci. USA, 100 (2003), pp. 11394–11399.

[12] J. Kneis, D. Mölle, S. Richter, and P. Rossmanith, *Divide-and-color*, in Graph-Theoretic Concepts in Computer Science (WG 2006), Lecture Notes in Comput. Sci. 4271, Springer, Berlin, 2006, pp. 58–67.

[13] I. Koutis, *A faster parameterized algorithm for set packing*, Inform. Process. Lett., 94 (2005), pp. 7–9.

[14] I. Koutis, *Faster algebraic algorithms for path and packing problems*, in Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP 2008), Part I, Lecture Notes in Comput. Sci. 5125, Springer, Berlin, 2008, pp. 575–586.

[15] L. Mathieson, E. Prieto, and P. Shaw, *Packing edge disjoint triangles: A parameterized view*, in Proceedings of the First International Workshop on Parameterized and Exact Computation (IWPEC 2004), Lecture Notes in Comput. Sci. 3162, Springer, Berlin, 2004, pp. 127–137.

[16] B. Monien, *How to find long paths efficiently*, Ann. Discrete Math., 25 (1985), pp. 239–254.

[17] M. Naor, L. Schulman, and A. Srinivasan, *Splitters and near-optimal derandomization*, in Proceedings of the 39th Annual Symposium on Foundations of Computer Science (FOCS 1995), IEEE Computer Society Press, Palo Alto, CA, 1995, pp. 182–190.

[18] A. Nilli, *Perfect hashing and probability*, Combin. Probab. Comput., 3 (1994), pp. 407–409.

[19] C. Papadimitriou and M. Yannakakis, *On limited nondeterminism and the complexity of the V-C dimension*, J. Comput. System Sci., 53 (1996), pp. 161–170.

[20] E. Prieto and C. Sloper, *Looking at the stars*, Theoret. Comput. Sci., 351 (2006), pp. 437–445.

[21] J. Scott, T. Ideker, R. Karp, and R. Sharan, *Efficient algorithms for detecting signaling pathways in protein interaction networks*, J. Comput. Biol., 13 (2006), pp. 133–144.

[22] V. Shoup, *A Computational Introduction to Number Theory and Algebra*, 2nd ed., Cambridge University Press, New York, 2008.

[23] R. Williams, *Finding paths of length k in $O^*(2^k)$ time*, Inform. Process. Lett., 109 (2009), pp. 315–318.