Improved Algorithms for Path, Matching, and Packing Problems*

Jianer Chen† Songjian Lu‡ Sing-Hoi Sze§ Fenghui Zhang‡

Abstract

Improved randomized and deterministic algorithms are presented for PATH, MATCHING, and PACKING problems. Our randomized algorithms are based on the divide-and-conquer technique, and improve previous best algorithms for these problems. For example, for the k-PATH problem, our randomized algorithm runs in time $O(4^k k^{3.42} m)$ and space $O(nk \log k + m)$, improving the previous best randomized algorithm for the problem that runs in time $O(5.44^k km)$ and space $O(2^k kn + m)$. To achieve improved deterministic algorithms, we study a number of previously proposed derandomization schemes, and also develop a new derandomization scheme. These studies result in a number of deterministic algorithms: one of time $O(4^{k+o(k)}m)$ for the k-PATH problem, one of time $O(2.80^{3k} kn \log^2 n)$ for the 3-D MATCHING problem, and one of time $O(4^{3k+o(k)}n)$ for the 3-SET PACKING problem. All these significantly improve previous best algorithms for the problems.

1 Introduction

This paper develops improved exact and parameterized algorithms for PATH, MATCHING, and PACKING problems that are NP-hard. This research direction has recently drawn considerable attention [2, 5, 7, 10, 11, 13, 15, 20].

The k-PATH problem (given a graph G and an integer k, either construct a simple path of k vertices in G or report that no such path exists) is closely related to a number of well-known NP-hard problems, such as the LONGEST PATH problem, the HAMILTONIAN PATH problem, and the TRAVELING SALESMAN problem. Earlier algorithms [3, 15] for the k-PATH problem have running time bounded by $O(2^k k! n^{O(1)})$. Papadimitriou and Yannakakis [18] studied a restricted version of the problem, the $(\log n)$-PATH problem, and conjectured that it can be solved in polynomial time. This conjecture was confirmed by Alon, Yuster, and Zwick [2], who presented for the k-PATH problem randomized and deterministic algorithms of running time $O(2^{O(k)} n^{O(1)})$. Very recently, the k-PATH problem has found applications in bioinformatics for detecting signaling pathways in protein interaction networks [20] and for biological subnetwork matchings [11].

Parameterized algorithms for MATCHING and PACKING problems were first studied in [6], where deterministic algorithms of running time $O(2^{O(k)}(3k)! n \log^4 n)$ were developed for the 3-D MATCHING problem (given a set S of triples and an integer k, either find a subset of k disjoint triples in S or report that no such subset exists) and the 3-SET PACKING problem (given a collection C of 3-sets and an integer k, either find a sub-collection of k disjoint 3-sets in C or report that no such sub-collection exists). The complexity upper bounds for these problems were subsequently improved to $O((5.7k)^k n)$ [5, 10]. Koutis [13] developed randomized algorithms of time $O(10.88^{3k} n^{O(1)})$ and space $O(2^{3k} + n)$, and improved deterministic algorithms of time $O(2^{O(k)} n^{O(1)})$ for these problems. The deterministic upper bound was further improved to $O((12.7D)^{3k} n^{O(1)})$ (where $D \geq 10.4$) by Fellows et al. [7].

Currently, the best randomized and deterministic algorithms for the k-PATH problem [2] and the MATCHING and SET PACKING problems [7, 13] are all based on the technique of *color coding* developed by Alon, Yuster, and Zwick [2]. Take the k-PATH problem as an example. We say that a simple path in a graph G is *properly colored* under a coloring of the vertices in G if no two vertices on the path are colored with the same color. The algorithms proposed in [2] proceed as follows. Suppose that there is a path P of k vertices in G, starting from a vertex v_0. To find the path P, first we color the vertices of the graph G using k colors so that the path P is properly colored. Then we use a (deterministic) dynamic programming algorithm, which for each vertex u records every possible color set C such that there is a properly colored simple path from v_0 to u that uses exactly the colors in the set C. Since there are at most 2^k different color sets, the dynamic programming algorithm runs in time $O(2^k km)$ and space $O(2^k kn + m)$.

Therefore, the critical step is how to construct a coloring for the graph G so that the path P is prop-

*This work was supported in part by the National Science Foundation under the Grants CCR-0311590 and CCF-0430683.

†Corresponding author, Department of Computer Science, Texas A&M University, College Station, TX 77843, Email: chen@cs.tamu.edu.

‡Department of Computer Science, Texas A&M University, College Station, TX 77843, Email: {sjlu,fhzhang}@cs.tamu.edu.

§Department of Computer Science and Department of Biochemistry & Biophysics, Texas A&M University, College Station, TX 77843, Email: shsze@cs.tamu.edu.

erly colored. Alon, Yuster, and Zwick [2] proposed two approaches to this problem. The first is a randomized algorithm of running time $O(e^k n)$ that produces $O(e^k)$ colorings for the graph G in which with high probability at least one coloring properly colors the path P. The second is a deterministic algorithm based on the hashing schemes studied by Fredman, Komlos, and Szemeredi [8] and Schmidt and Siegel [19], which constructs a set of $O(c^k n^{O(1)})$ colorings for the graph G in which at least one colors the path P properly, where c is a large constant. This, plus the above dynamic programming algorithm, gives for the k-PATH problem a randomized algorithm of running time $O((2e)^k n^{O(1)}) = O(5.44^k n^{O(1)})$ and space $O(2^k kn+m)$ and a deterministic algorithm of running time $O((2c)^k n^{O(1)})$. The current best randomized and deterministic algorithms for MATCHING and SET PACKING [13, 7] follow the same principle: first color the elements so that no two elements in the subset of interest are colored with the same color, then apply a deterministic algorithm (e.g., dynamic programming) on the set of colored elements to search for the subset.

This method is of great theoretical importance. In particular, it confirms Papadimitriou and Yannakakis's conjecture that the $(\log n)$-PATH problem can be solved in polynomial time. On the other hand, both the time and space complexity of the algorithms are quite high. From a practical point of view, it is necessary to further significantly improve the running time of the algorithms to make them practically useful for moderate values of k. Moreover, the space complexity of all the randomized algorithms described above for PATH, MATCHING, and PACKING is exponential in k, which is also remarkable.

In this paper, we develop improved randomized and deterministic algorithms for the PATH, MATCHING, and PACKING problems. Our first constribution is a randomized divide-and-conquer method. Roughly speaking, suppose we are looking for a subset S_k of k elements in a large set S. We first randomly partition the set S into two parts, then recursively look for a subset of $k/2$ elements in each part. This simple method leads directly to improved randomized algorithms. For the k-PATH problem, this new method gives a randomized algorithm of time $O(4^k k^{3.42} m)$ and space $O(nk \log k + m)$,[1] improving the previous best randomized algorithm for the problem of time $O(5.44^k km)$ and space $O(2^k kn + m)$ [2]. For the 3-D MATCHING and 3-SET PACKING problems, the method gives randomized algorithms of time $O(2.52^{3k} n)$ and space $O(nk \log k + m)$, improving the previous best randomized algorithms for the problems of time $O(10.88^{3k} n^{O(1)})$ and space $O(2^{3k} + m)$ [13].

[1] We heard recently that Kneis et al. [12] have independently developed a similar randomized algorithm for the k-PATH problem, with running time similar to that of ours presented in this paper.

To achieve improved deterministic algorithms, we study a number of previously proposed derandomization schemes, including the (n, k)-universal sets studied in [16], and the (n, k)-families of perfect hashing functions studied in [2, 16]. Using the derandomization scheme based on (n, k)-universal sets, we derive a deterministic algorithm of running time $O(4^{k+o(k)} m)$ for the k-PATH problem, and a deterministic algorithm of running time $O(4^{3k+o(k)} n)$ for the 3-SET PACKING problem. We also develop a new (n, k)-family of perfect hashing functions, which, plus other techniques, gives a deterministic algorithm of running time $O(2.80^{3k} kn \log^2 n)$ for the 3-D MATCHING problem. All these results significantly improve previous best algorithms for the problems.

The paper is organized as follows. Section 2 presents improved randomized and deterministic algorithms based on the divide-and-conquer technique for the PATH, MATCHING, and PACKING problems. In section 3, we develop a new family of perfect hashing functions that can be constructed more efficiently in terms of both time and space complexity. Using this new family, section 4 develops a further improved deterministic algorithm for the 3-D MATCHING problem.

2 Improved algorithms via divide-and-conquer

We start with a group of new randomized algorithms based on the divide-and-conquer method. These algorithms are improvements over previous randomized algorithms for a number of PATH, MATCHING, and PACKING problems. To make our discussion more specific, we will describe the method in detail based on the k-PATH problem. We then explain briefly how the method is applied to MATCHING and PACKING problems. Finally, we discuss how these algorithms are derandomized to achieve improved deterministic algorithms for the problems. Throughout this paper, we will denote by $e = 2.718\cdots$ the base of the natural logarithm.

2.1 The randomized algorithms The randomized algorithm for k-PATH is given in Figure 1. A simple path in a graph G is a (u, k)-*path* if it contains exactly k vertices and if one end of the path is u. In particular, a $(u, 1)$-path consists of a single vertex u. When the vertex u is irrelevant, a (u, k)-path will be simply called a k-*path*. Our algorithm **find-paths**(P', G', k) on a set P' of k'-paths and a subgraph G' in G (where no vertex in G' is on any path in P') returns a set P of paths, each is a concatenation of a k'-path in P' and a k-path in G' (if no such paths exist, the algorithm returns an empty set). In particular, the algorithm **find-paths**$(\emptyset, G', k)$ returns a set of k-paths in the graph G'.

LEMMA 2.1. *For all* $k \geq 2$, $\lceil \log k \rceil = \lceil \log(\lceil \frac{k}{2} \rceil) \rceil + 1$.

find-paths(P', G', k)
input: P' a set of k'-paths and G' a subgraph in G;
G' contains no vertex in P', an integer $k \geq 1$;
output: a set P of paths, each is a concatenation of a k'-path in P' and a k-path in G';

1. $P = \emptyset$;
2. **if** $k = 1$ **then**
   if $P' = \emptyset$ **then return** all 1-paths in G';
   else for each (u, k')-path p in P' and each edge (u, w) in G, where w is in G' **do**
   concatenate p and w to make a $(w, k'+1)$-path p';
   add p' to P if no $(w, k'+1)$-path is in P;
   return P;
3. **loop** $2.51 \cdot 2^k$ times **do**
3.1. randomly partition the vertices in G' into two parts V_L and V_R;
3.2. let G_L and G_R be the subgraphs induced by V_L and V_R, respectively;
3.3. $P_L =$ **find-paths**$(P', G_L, \lceil k/2 \rceil)$;
3.4. **if** $P_L \neq \emptyset$ **then**
3.5. $P_R =$ **find-paths**$(P_L, G_R, k - \lceil k/2 \rceil)$;
3.6. **for** each $(u, k'+k)$-path p in P_R **do**
3.7. add p to P if no $(u, k'+k)$-path is in P;
4. **return** P.

Figure 1: A divide-and-conquer algorithm for k-PATH

THEOREM 2.1. *On a graph $G = (V, E)$ with n vertices and m edges and an integer $k \geq 1$, if G contains a (u, k)-path for a vertex u, then with probability larger than $1 - 1/e > 0.632$, the set P returned by the algorithm* **find-paths**$(\emptyset, G, k)$ *contains a (u, k)-path. The algorithm* **find-paths**$(\emptyset, G, k)$ *runs in time $O(4^k k^{3.42} m)$ and in space $O(nk \log k + m)$.*

Proof. To prove the first part, we prove the following claims using induction on k:

1. If $P' = \emptyset$ and G' has a (u, k)-path, then with probability larger than $1 - 1/e$, the set P returned by the algorithm **find-paths**(P', G', k) includes a (u, k)-path.
2. If $P' \neq \emptyset$ and G' has a (u, k)-path whose other end is connected to an end vertex of a path in P', then with probability larger than $1 - 1/e$, the set P returned by the algorithm **find-paths**(P', G', k) contains a $(u, k'+k)$-path.

The claims are obviously true for $k = 1$. Let $k > 1$. First consider the case when $P' = \emptyset$. Suppose that

$$[u_1, u_2, \ldots, u_{k_1}, u_{k_1+1}, \ldots, u_k]$$

is a (u_k, k)-path in G', where $k_1 = \lceil k/2 \rceil$. Then with probability $1/2^k$, step 3.1 of the algorithm puts vertices $u_1, u_2, \ldots, u_{k_1}$ into V_L, and vertices $u_{k_1+1}, \ldots, u_k$ into V_R. If this is the case, then the graph G_L contains the (u_{k_1}, k_1)-path $[u_1, \ldots, u_{k_1}]$, and the graph G_R contains the $(u_k, k - k_1)$-path $[u_{k_1+1}, \ldots, u_k]$. By the inductive hypothesis, with probability larger than $1 - 1/e$, P_L obtained from step 3.3 includes a (u_{k_1}, k_1)-path. The $(u_k, k-k_1)$-path $[u_{k_1+1}, \ldots, u_k]$ in G_R has its other end u_{k_1+1} connected to the (u_{k_1}, k_1)-path in P_L. Therefore with probability larger than $1-1/e$, P_R obtained in step 3.5 contains a path of length $k_1 + (k - k_1) = k$ that ends with u_k, i.e., a (u_k, k)-path. Therefore in each loop of step 3, the probability ρ that a (u_k, k)-path is added to the set P is larger than

$$\frac{(1-1/e)^2}{2^k} > \frac{0.6322}{2^k} > \frac{1}{2.51 \cdot 2^k}.$$

In the case when $P' \neq \emptyset$, we follow the same argument as before except that we require that the (u_k, k)-path in G' has its other end connected to the end of a k'-path in P'. So P_L contains a $(u_{k_1}, k' + k_1)$-path p that is a concatenation of a k'-path in P' and a k_1-path in G_L, and P_R contains a $(u_k, k' + k)$-path that is a concatenation of a $(k' + k_1)$-path in P_L and a $(k - k_1)$-path in G_R.

Since step 3 of the algorithm loops $2.51 \cdot 2^k$ times, the overall probability that the algorithm returns a set of paths that contains a (u_k, k)-path (when $P' = \emptyset$) or a $(u_k, k' + k)$-path (when $P' \neq \emptyset$) is

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

This proves the first part of the theorem.

To analyze the time complexity, let $T(k)$ be the running time of the algorithm **find-paths**(P', G', k). Without loss of generality, we assume that $m \geq n$. From the algorithm, we get the following recurrence relation:

$$T(k) = 2.51 \cdot 2^k [cm + T(\lceil k/2 \rceil) + T(k - \lceil k/2 \rceil)],$$

where $c > 0$ is a constant. We claim that for all $k > 0$,

$$T(k) \leq c \cdot (10.7)^{\lceil \log k \rceil} 2^{2k} m, \tag{2.1}$$

and we prove it by induction on k. Obviously $T(1) \leq cm$ if c is sufficiently large, so inequality (2.1) holds for $k = 1$. Let $k > 1$, then

$$\begin{aligned}
T(k) &= 2.51 \cdot 2^k \left(cm + T(\lceil k/2 \rceil) + T(k - \lceil k/2 \rceil)\right) \\
&\leq 2.51 \cdot 2^k \left(cm + 2c \cdot (10.7)^{\lceil \log \lceil k/2 \rceil \rceil} m 2^{2\lceil k/2 \rceil}\right) \\
&\leq 2.51 \cdot 2^k \left(cm + \tfrac{2c}{10.7} \cdot (10.7)^{\lceil \log \lceil k/2 \rceil \rceil + 1} m 2^{k+1}\right) \\
&= c \cdot (10.7)^{\lceil \log k \rceil} 2^{2k} m \cdot 2.51 \left(\tfrac{1}{10.7^{\lceil \log k \rceil} 2^k} + \tfrac{4}{10.7}\right) \\
&\leq c \cdot (10.7)^{\lceil \log k \rceil} 2^{2k} m \cdot 2.51 \left(\tfrac{1}{10.7 \cdot 4} + \tfrac{4}{10.7}\right) \\
&< c \cdot (10.7)^{\lceil \log k \rceil} 2^{2k} m.
\end{aligned}$$

Here in the second step of the above derivation, we have used $k - \lceil k/2 \rceil \leq \lceil k/2 \rceil$. In the third step, we have

used $2\lceil k/2 \rceil \leq k+1$, and in the fourth step we have used Lemma 2.1. Thus the running time $T(k)$ of the algorithm **find-paths**$(\emptyset, G, k)$ is $O((10.7)^{\lceil \log k \rceil} 2^{2k} m) = O(4^k k^{3.42} m)$.

In terms of the space complexity, each recursive call to the algorithm **find-paths** uses $O(nk)$ space (mainly for the sets P_L, P_R, and P). Since on a graph G and an integer k, the recursive depth of the algorithm is $O(\log k)$, we conclude that the space complexity of the algorithm **find-paths**$(\emptyset, G, k)$ is $O(nk \log k + m)$.

To obtain a randomized algorithm solving the k-PATH problem with a required error bound, we simply run the algorithm in Theorem 2.1 sufficiently many times. For example, to achieve an error bound of 0.0001, we can run the algorithm in Theorem 2.1 t times, where t satisfies $(1/e)^t \leq 0.0001$ (e.g., $t = 10$).

COROLLARY 2.1. *There is a randomized algorithm of time $O(4^k k^{3.42} m)$ and space $O(nk \log k + m)$ that solves the k-PATH problem with arbitrarily small error bound.*

REMARK 1. Recently, Kneis et al. [12] have independently developed a similar randomized algorithm for the k-PATH problem. The running time of their algorithm is similar to that in Corollary 2.1.

REMARK 2. It seems that we have to be more careful when we analyze an exponential time algorithm based on the divide-and-conquer method. Certain common techniques from traditional algorithm analysis do not seem to be directly applicable. For example, we cannot simply assume that the parameter k is a power of 2 since the extension from this special case to the case for general k does not seem to give the same complexity bound. In fact, when k is a power of 2, it is quite trivial to verify (by induction) that $T(k) \leq O(4^k k^{2.52} m)$. However, it seems not easy to extend this bound to the case for general k.

We compare our algorithm in Corollary 2.1 with previously known algorithms for the k-PATH problem. To our knowledge, there are two kinds of randomized algorithms for the k-PATH 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 followed by dynamic programming to search for a k-path 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 only need to know whether the graph has a k-path, a slight modification of our algorithm can further reduce the space complexity to $O(n \log k + m)$.

The above randomized divide-and-conquer method can also be used to develop improved algorithms for MATCHING and PACKING problems. Consider the 3-D MATCHING problem (given a set S of triples and an integer k, either find a subset S_k of k disjoint triples or report that no such subset exists). In the case when such a subset S_k exists, let A_k be the set of $3k$ symbols in the triples in S_k. With probability $\binom{k}{k/2}/2^{3k} = O(1/(2^{2k}\sqrt{k}))$, we can partition the symbols in A_k into two subsets A'_k and A''_k such that A'_k contains $3k/2$ symbols in $k/2$ triples in S_k and A''_k contains $3k/2$ symbols in the other $k/2$ triples in S_k. The set S of triples can be partitioned into two subsets S' and S'' in terms of A'_k and A''_k, and a subset of $k/2$ triples is searched recursively in each of the sets S' and S''. An analysis similar to that in Theorem 2.1 shows that this algorithm runs in time $O(4^{2k+o(k)}n) = O(2.52^{3k}n)$ and space $O(nk \log k + m)$ and finds the subset of k triples with high probability. It is straightforward to modify this approach to obtain an algorithm of the same time and space complexity for the 3-SET PACKING problem (given a collection of 3-sets and an integer k, either find a sub-collection of k disjoint 3-sets or report that no such sub-collection exists).

THEOREM 2.2. *The* 3-D MATCHING *and* 3-SET PACKING *problems can be solved by randomized algorithms in time $O(2.52^{3k}n)$ and space $O(nk \log k + m)$.*

The previous best randomized algorithms for the problems [13] take time $O(10.88^{3k} n^{O(1)})$ and space $O(2^{3k} + m)$, where the space is exponential in k.

2.2 Derandomization We discuss how the randomized algorithms presented in the previous subsection are derandomized. Kneis et al. [12] suggested a derandomization process for their randomized algorithm for the k-PATH problem (which is essentially the same as our algorithm given in Figure 1) based on the construction of almost k-wise independent random variables developed in [1]. Their derandomization process results in a deterministic algorithm of running time $O(16^k n^{O(1)})$ for the k-PATH problem. In this subsection, we describe an improved derandomization process based on the construction of (n, k)-universal sets studied in [16].

Let $s = s_1 s_2 \cdots s_n$ be an n-bit binary string, and let π be a subset of k elements in the index set $\{1, 2, \ldots, n\}$. We will denote by $\pi(s)$ the k-bit binary string obtained from s by deleting all bits s_i where $i \notin \pi$.

DEFINITION 2.1. An (n, k)-*universal set* T is a set of n-bit binary strings such that for every subset π of k

elements in $\{1, 2, \ldots, n\}$, the collection $\{\pi(s) \mid s \in T\}$ contains all 2^k k-bit binary strings. The *size* of the (n, k)-universal set T is the number of strings in T.

PROPOSITION 2.1. ([16]) *There is a deterministic algorithm of running time $O(2^k k^{O(\log k)} n \log n)$ that constructs an (n, k)-universal set of size $2^k k^{O(\log k)} \log n$.*

We explain how Proposition 2.1 is used to achieve a deterministic algorithm for the k-PATH problem. Consider the algorithm **find-path**(P', G', k) in Figure 1. Suppose that the graph G' has n vertices v_1, ..., v_n. First, we construct an (n, k)-universal set T of size $2^k k^{O(\log k)} \log n$ (this can be done in time $O(2^k k^{O(\log k)} n \log n)$). Then we call the algorithm **find-path**(P', G', k) but replace step 3 and step 3.1 in the algorithm by the following steps:

3. **for** each n-bit binary string $s = s_1 s_2 \cdots s_n$ in T **do**
3.1. let $V_L = \{v_i \mid s_i = 0\}$ and $V_R = \{v_i \mid s_i = 1\}$;

Note that this replacement makes the algorithm **find-path**(P', G', k) become deterministic. Moreover, suppose that there is a k-path P in the graph G'. Then, since T is an (n, k)-universal set, one s of the n-bit binary strings in T picked in step 3 in the algorithm will make step 3.1 to include the first $\lceil k/2 \rceil$ vertices on P in the set V_L and the last $k - \lceil k/2 \rceil$ vertices on P in the set V_R. Now, the reasoning goes very similar to that of Theorem 2.1. Using induction on k, we can prove that this deterministic algorithm correctly returns a k-path if such a path exists in the input graph. Finally, completely similarly to that in Theorem 2.1, we can derive that the running time of this deterministic algorithm is bounded by $O(4^{k+o(k)} m)$.

THEOREM 2.3. *There is an $O(4^{k+o(k)} m)$ time deterministic algorithm that solves the k-PATH problem.*

Similarly, using $(n, 3k)$-universal sets, we can derandomize the randomized algorithms for the 3D-MATCHING and 3-SET PACKING problems.

THEOREM 2.4. *There are deterministic algorithms of running time $O(4^{3k+o(k)} n)$ that solve the* 3D-MATCHING *and* 3-SET PACKING *problems.*

3 On perfect hashing function families

To obtain further improved deterministic algorithms for MATCHING and PACKING problems, we investigate the method of *color coding* proposed by Alon, Yuster, and Zwick [2] and develop a new (n, k)-family of perfect hashing functions. The previous best explicit construction for perfect hashing function families was suggested in [2] based on the construction of [19], which gives an (n, k)-family of perfect hashing functions of size at least $\Omega(11^k \log n)$ (this bound was not made explicit in the paper). Another construction of (n, k)-families of perfect hashing functions of size $O(e^k k^{O(\log k)} \log n)$ was described in [16], which depends on the existence of a probability space that we were not able to verify. Moreover, it seems that the construction given in [16] requires super-polynomial space.

We will present in this section a new (n, k)-family of perfect hashing functions of size $O(6.4^k \log^2 n)$. Moreover, the representation of our (n, k)-family of perfect hashing functions only requires polynomial space. This new (n, k)-family of perfect hashing functions directly induces improved algorithms for a number of MATCHING and PACKING problems.

3.1 Preliminaries Let S be a set and let W be a subset of S. A function f on S is *injective* from W if for any two different elements x and y in W, $f(x) \neq f(y)$. For each integer m, denote by Z_m the set $\{0, 1, \ldots, m-1\}$. In particular, if m is a prime number, then Z_m is a field under the addition and multiplication modulo m.

DEFINITION 3.1. A *k-coloring* of the set Z_n is a function from Z_n to Z_k. A collection $\mathcal{F}$ of k-colorings of Z_n is an *(n, k)-family of perfect hashing functions* if for any subset W of k elements in Z_n, there is a k-coloring in $\mathcal{F}$ that is injective from W. The *size* of the family $\mathcal{F}$ is the number of k-colorings in $\mathcal{F}$.

An (n, k)-family of perfect hashing functions can be constructed from a (k^2, k)-family of perfect hashing functions, as described in the following, which was implicitly proved in [8].

THEOREM 3.1. ([8]) *If there is a (k^2, k)-family of perfect hashing functions of size r, then there is an (n, k)-family of perfect hashing functions of size bounded by $k^4 r \log^2 n$.*

If we denote by $\tau(n, k)$ the minimum size of an (n, k)-family of perfect hashing functions, then from Theorem 3.1, we get immediately:

COROLLARY 3.1. $\tau(n, k) \leq \tau(k^2, k) \cdot k^4 \log^2 n$.

Therefore, the values $\tau(n, k)$ and $\tau(k^2, k)$ differ by at most a factor of $k^4 \log^2 n$. This "kernelization" result enables us to concentrate on the study of the case where $n = k^2$. Moreover, we should point out that a lower bound $\Omega(e^k \log n / \sqrt{k})$ for the value $\tau(n, k)$ has also been derived [17].

3.2 Perfect hashing functions for small k We first study the properties of (n,k)-family of perfect hashing functions, from which we will be able to derive upper bounds for the value $\tau(n,k)$ for small n and k. The proofs of the results in this subsection will be presented in the full version of the paper.

LEMMA 3.1. *For any integers n and k, $n \geq k$, there is an (n,k)-family of perfect hashing functions of size $\binom{n}{k}$.*

For the case $k \leq 2$ and $k = n$, we have

LEMMA 3.2. *For any $n \geq 2$, $\tau(n,0) \leq 1$, $\tau(n,1) = 1$, $\tau(n,n) = 1$, and $\tau(n,2) \leq \lceil \log n \rceil$.*

For a general k, we have the following result.

LEMMA 3.3. *If $n = n_1 + \cdots + n_r$, where $n_j \geq 1$ for all $1 \leq j \leq r$, then*

$$\tau(n,k) \leq \sum_{0 \leq k_1 \leq n_1, \ldots, 0 \leq k_r \leq n_r}^{k_1+\cdots+k_r=k} \left(\frac{\tau(|k_j \leq 1|, |k_j = 1|)}{\binom{|k_j \leq 1|}{|k_j = 1|}} \prod_{k_j \geq 2} \tau(n_j, k_j) \right),$$

where $|k_j \leq 1|$ and $|k_j = 1|$ denote the numbers of k_j's in the list $[k_1, \ldots, k_r]$ such that $k_j \leq 1$ and $k_j = 1$, respectively.

By Lemma 3.2 and Lemma 3.3, for small values of n and k, we can derive specific upper bounds $\tau_0(n,k)$ for the values $\tau(n,k)$ and construct (n,k)-families of perfect hashing functions. We first did this for very small values of n and k. Then using a computer program and based on Lemma 3.3, we also did this for larger values of n and k. The values $\tau_0(n,k)$ for these small n and k enabled us to prove the following lemma.

LEMMA 3.4. *There is a collection of $(k(k-1),k)$-family of perfect hashing functions $\mathcal{F}_{k(k-1)}$ of size $\tau_0(k(k-1),k)$, for $k \geq 2$, such that for any list of non-negative integers $[k_1, k_2, \ldots, k_r]$ satisfying $k_1 + \cdots + k_r = k$ and $\sum_{j=1}^{r} k_j(k_j - 1) \leq 4k$, the following inequality holds: $\prod_{k_j \geq 2} \tau_0(k_j(k_j-1), k_j) \leq 2.4142^k$.*

3.3 A new family of perfect hashing functions We first consider the case where k is divisible by 4 and let $k' = k/4 - 1$. The case for general k will be handled after the discussion for this special case. According to Theorem 3.1, we can concentrate on the case $n = k^2$. Therefore, throughout the rest of this section, we assume that $n = k^2$.

Let p be a prime number satisfying $n \leq p < 2n$ (such a prime number exists by Bertrand's Conjecture [9]). The prime number p can be obtained in time $O(n\sqrt{n}) = O(k^2)$ using the straightforward primality testing algorithm.

We first present a k-coloring algorithm for the set Z_n. Our k-coloring algorithm is associated with a set of parameters satisfying the following properties:

C1 A pair of integers (a,b), where $0 < a \leq p-1$, $0 \leq b \leq p-1$;

C2 A list $C = [c_0, c_1, \ldots, c_{k'}]$ of non-negative integers, where $k' = k/4 - 1$, $\sum_{j=0}^{k'} c_j = k$, and $\sum_{j=0}^{k'} c_j(c_j - 1) \leq 4k$. Let $C_{>1}$ be the sublist of C by removing all $c_j \leq 1$;

C3 A list $L = [(a_1,b_1), (a_2,b_2), \ldots, (a_r,b_r)]$ of pairs of integers, where $0 < a_i \leq p-1$, $0 \leq b_i \leq p-1$, and $r \leq \log |C_{>1}|$;

C4 A mapping from the elements in the list $C_{>1}$ to the elements in the list L such that at least half of the c_j's in $C_{>1}$ are mapped to (a_1,b_1), at least half of the c_j's that are not mapped to (a_1,b_1) are mapped to (a_2,b_2), at least half of the c_j's that are not mapped to (a_1,b_1) and (a_2,b_2) are mapped to (a_3,b_3), and so on.

C5 A list of colorings $[F_{c_0}, F_{c_1}, \ldots, F_{c_{k'}}]$, where for $c_j > 1$, F_{c_j} is a c_j-coloring of the set $Z_{c_j(c_j-1)}$ taken from the $(c_j(c_j-1), c_j)$-family $\mathcal{F}_{c_j(c_j-1)}$ of perfect hashing functions given in Lemma 3.4 (for $c_j \leq 1$, F_{c_j} is irrelevant).

We also define a function as follows. For three given integers s, a, b, where $1 < s < n$, $0 < a \leq p-1$, and $0 \leq b \leq p-1$, define a function $\phi_{a,b,s}$ from the set Z_n to the set Z_s by

$$\phi_{a,b,s}(x) = ((ax+b) \bmod p) \bmod s. \tag{3.2}$$

Our k-coloring algorithm is given in Figure 2. Since $\sum_{j=0}^{k'} c_j = k$, it is easy to verify that the algorithm **Coloring** produces a k-coloring for the set Z_n.

For each different set of parameters satisfying conditions **C1**–**C5**, the algorithm **Coloring** produces a k-coloring for the set Z_n. We first consider the number of different combinations of these parameters satisfying conditions **C1**–**C5**.

THEOREM 3.2. *The total number of combinations of the parameters satisfying conditions* **C1**–**C5** *is bounded by $O(6.383^k k^{4\log k - 4})$, and these combinations can be enumerated systematically.*

Proof. There are $p^2 = O(n^2) = O(k^4)$ pairs of integers (a,b) satisfying condition **C1**.

Coloring
input: parameters as specified in **C1**–**C5**;
output: a k-coloring of the set Z_n;

1. **for** $j = 0$ **to** $k' = k/4 - 1$ **do**
   $U_j = \{x \mid x \in Z_n,\ \phi_{a,b,k/4}(x) = j\}$;
2. **for** each U_j such that $c_j > 1$, suppose that c_j is mapped to (a_i, b_i) **do**
   for $t = 0$ **to** $c_j(c_j - 1) - 1$ **do**
   $U_{j,t} = \{x \mid x \in U_j,\ \phi_{a_i,b_i,c_j(c_j-1)}(x) = t\}$;
   create c_j new colors $\tau_{j,0}, \tau_{j,1}, \ldots, \tau_{j,c_j-1}$;
   assign all elements in $U_{j,t}$ with color $\tau_{j,s}$ if the c_j-coloring F_{c_j} for $Z_{c_j(c_j-1)}$ assigns color s to the element t;
3. for each $c_j = 1$, create a new color τ_j and assign all elements in U_j the color τ_j;
4. assign all elements in $\bigcup_{c_j=0} U_j$ arbitrarily using the colors created in steps 2–3.

Figure 2: A coloring algorithm

Consider condition **C2**. We represent each list $C = [c_0, \ldots, c_{k'}]$ satisfying condition **C2** using a single binary string B_C of length $5k/4 - 1$ in which there are exactly $k/4 - 1$ 0-bits. The $k/4 - 1$ 0-bits in B_C divide B_C into $k/4$ "segments" such that the jth segment contains exactly c_j 1-bits (in particular, the segment between two consecutive 0's in B_C corresponds to a $c_j = 0$). It is easy to verify that any list C satisfying condition **C2** is uniquely represented by such a binary string B_C. Note that the number of binary strings of length $5k/4 - 1$ with exactly $k/4 - 1$ 0-bits is equal to $\binom{5k/4-1}{k/4-1} \leq 1.8692^k$. We conclude that the total number of different lists satisfying condition **C2** is bounded by 1.8692^k. Note that all these lists can be systematically enumerated based on the binary string representation described above.

Since $r \leq \log(|C_{>1}|) \leq \log(k/4) = \log k - 2$, there are at most $\log k - 2$ pairs in each list L satisfying condition **C3**. By condition **C3**, there are $p^2 = O(k^4)$ possible pairs for each (a_i, b_i). Therefore, the total number of lists L satisfying condition **C3** is bounded by $O(k^{4 \log k - 8})$.

Now we discuss when a list $C = [c_0, \ldots, c_{k'}]$ satisfying condition **C2** and a list $L = [(a_1, b_1), \ldots, (a_r, b_r)]$ satisfying condition **C3** are given, how many different mappings from $C_{>1}$ to L can there be that satisfy condition **C4**. Let $q = |C_{>1}| \leq k/4$. We use a binary string $A_{>1}$ to represent a mapping from $C_{>1}$ to L. The binary string $A_{>1}$ has q 0-bits, which divide $A_{>1}$ into q segments, each starting with a 0-bit. For each j, the jth segment of the form 01^{i-1} in $A_{>1}$ represents the mapping from the jth element in $C_{>1}$ to the integer pair (a_i, b_i) in L. By Condition **C4**, at least half of the segments in $A_{>1}$ have no 1-bit, at least half of the remaining segments in $A_{>1}$ have the form 01, and at least half of the remaining segments that are not of the form 0 or 01 in $A_{>1}$ have the form 011, and so on. Therefore, the length of the binary string $A_{>1}$ is bounded by

$$\frac{q}{2} + 2\frac{q}{2^2} + 3\frac{q}{2^3} + \cdots < 2q \leq \frac{k}{2}.$$

In consequence, the number of different mappings from $C_{>1}$ to L satisfying condition **C4** is bounded by $2^{k/2} = 1.4143^k$.

Finally, for each $c_j > 1$, the c_j-coloring F_{c_j} in the list satisfying condition **C5** is from the $(c_j(c_j - 1), c_j)$-family $\mathcal{F}_{c_j(c_j-1)}$ of perfect hashing functions given in Lemma 3.4, whose size is $\tau_0(c_j(c_j - 1), c_j)$. The total number of lists of colorings satisfying condition **C5** is bounded by $\prod_{c_j \geq 2} \tau_0(c_j(c_j - 1), c_j)$. By condition **C2**, the numbers c_j satisfy $\sum_{j=0}^{k'} c_j = k$ and $\sum_{j=0}^{k'} c_j(c_j - 1) \leq 4k$. Thus by Lemma 3.4,

$$\prod_{c_j \geq 2} \tau_0(c_j(c_j - 1), c_j) \leq 2.4142^k.$$

Combining all these results, we conclude that the total number of combinations of the parameters satisfying conditions **C1**–**C5** is bounded by

$$\begin{aligned} &O(k^4) \cdot 1.8692^k \cdot O(k^{4 \log k - 8}) \cdot 1.4143^k \cdot 2.4142^k \\ &\quad = O(6.383^k k^{4 \log k - 4}). \end{aligned}$$

Moreover, from the above discussion, these combinations can be enumerated systematically.

COROLLARY 3.2. *Running the algorithm* **Coloring** *in Figure 2 over all possible parameters satisfying conditions* **C1**–**C5** *gives a collection $\mathcal{F}$ of $O(6.383^k k^{4 \log k - 4})$ k-colorings for the set Z_n.*

We show that the collection $\mathcal{F}$ of k-colorings in Corollary 3.2 makes an (n, k)-family of perfect hashing functions.

Consider the following two sets of ordered pairs of integers:

$$F_1(p) = \{(a, b) \mid 0 < a \leq p - 1 \text{ and } 0 \leq b \leq p - 1\},$$
$$F_2(p) = \{(r, q) \mid 0 \leq r, q \leq p - 1 \text{ and } r \neq q\}.$$

Fix two distinct integers x and y, $0 \leq x, y \leq p - 1$, we construct a mapping as follows:

$$\pi :\ (a, b) \longrightarrow ((ax + b) \bmod p,\ (ay + b) \bmod p).$$

LEMMA 3.5. *For any two integers x and y such that $0 \leq x, y \leq p - 1$ and $x \neq y$, the mapping π is a one-to-one mapping from $F_1(p)$ to $F_2(p)$.*

For a given integer s, $1 < s < n$, and an ordered pair (a, b) in $F_1(p)$, recall the function $\phi_{a,b,s}$ defined in

(3.2) from the set Z_n to the set Z_s. For each subset W of the set Z_n and for each integer j, $0 \le j \le s-1$, denote by $B(a,b,s,W,j)$ the number of integers x in W such that $\phi_{a,b,s}(x) = j$. We have the following lemma.

LEMMA 3.6. *Suppose $p \bmod s = h$. Then for every subset W of k elements in Z_n, we have*

$$\begin{aligned} &\textstyle\sum_{(a,b)\in F_1(p)} \sum_{j=0}^{s-1} \binom{B(a,b,s,W,j)}{2} \\ &= \tfrac{k(k-1)(p-h)(p-(s-h))}{2s}. \end{aligned} \tag{3.3}$$

COROLLARY 3.3. *Let $1 < s < n$. For each subset W of k elements in Z_n, there is an ordered pair (a,b) in $F_1(p)$ such that*

$$\sum_{j=0}^{s-1} \binom{B(a,b,s,W,j)}{2} < \frac{k(k-1)}{2s}.$$

We remark that Corollary 3.3 gives a significant improvement over the bound given by Fredman, Komlos, and Szemeredi [8], which was used to implement the (n,k)-family of perfect hashing functions suggested by Alon, Yuster, and Zwick [2]. In particular, the bound derived in [8] uses a hashing function $\psi_{a,s}$ defined by

$$\psi_{a,s}(x) = (ax \bmod p) \bmod s,$$

and the corresponding bound is k^2/s. On the other hand, our bound is derived based on the hashing function $\phi_{a,b,s}$. The hashing function $\phi_{a,b,s}$ has been studied by Carter and Wegman for other purposes [4]. Roughly speaking, the value in (3.3) measures the collision (i.e., the squared sum of "radii") of a hashing function. It has been shown [4] that no hashing function can significantly improve Corollary 3.3. We will show that the bound improvement from k^2/s to $k(k-1)/(2s)$ will induce a very significant improvement on the upper bound for the size $\tau(n,k)$ of (n,k)-families of hashing functions.

Now we are ready to show that the collection $\mathcal{F}$ in Corollary 3.2 makes an (n,k)-familiy of hashing functions. For this, we need to show that for every subset W of k elements in Z_n, there is a selection of the parameters satisfying conditions **C1**–**C5**, on which the algorithm **Coloring** in Figure 2 produces a k-coloring that is injective from W.

LEMMA 3.7. *For a subset W of k elements in Z_n, there is a pair (a',b') in $F_1(p)$ such that*

$$\sum_{j=0}^{k/4-1} B(a',b',k/4,W,j)(B(a',b',k/4,W,j)-1) < 4k.$$

COROLLARY 3.4. *For a subset W of k elements in Z_n, there is a pair (a',b') satisfying condition* **C1**, *such that if we let $W_j = \{x \mid \phi_{a',b',k/4}(x) = j\}$ and $c'_j = |W_j|$ for all $0 \le j \le k' = k/4-1$, then the list $C' = [c'_0, \ldots, c'_{k'}]$ satisfies condition* **C2**.

According to Corollary 3.4, there is a pair (a',b') satisfying condition **C1**, on which each set U_j constructed in step 1 of the algorithm **Coloring** contains c'_j elements in the subset W for $0 \le j \le k'$, and the list $C' = [c'_0, \ldots, c'_{k'}]$ satisfies condition **C2**.

Let $\mathcal{W}$ be a collection of some of the subsets W_j with $c'_j > 1$, as given in Corollary 3.4 ($\mathcal{W}$ may not necessarily contain all such subsets). Then we have the following lemma.

LEMMA 3.8. *Let $\mathcal{W}$ be any collection of subsets W_j of W as given in Corollary 3.4 where $c'_j = |W_j| > 1$. Then there is a pair (a'',b'') satisfying condition* **C1** *such that for at least one half of the subsets W_j in $\mathcal{W}$, the function $\phi_{a'',b'',c'_j(c'_j-1)}$ is injective from W_j to $Z_{c'_j(c'_j-1)}$.*

COROLLARY 3.5. *Let $\mathcal{W}_{>1}$ be the collection of all subsets W_j in Corollary 3.4 with $c'_j = |W_j| > 1$. Then there is a list $L' = [(a'_1,b'_1), \ldots, (a'_r,b'_r)]$ satisfying condition* **C3** *such that for at least one half of the subsets W_j in $\mathcal{W}_{>1}$, the function $\phi_{a'_1,b'_1,c'_j(c'_j-1)}$ is injective from W_j to $Z_{c'_j(c'_j-1)}$; for at least one half of the remaining subsets W_j in $\mathcal{W}_{>1}$, the function $\phi_{a'_2,b'_2,c'_j(c'_j-1)}$ is injective from W_j to $Z_{c'_j(c'_j-1)}$; and for at least one half of the remaining subsets W_j in $\mathcal{W}_{>1}$, the function $\phi_{a'_3,b'_3,c'_j(c'_j-1)}$ is injective from W_j to $Z_{c'_j(c'_j-1)}$; and so on.*

COROLLARY 3.6. *Let W_j, $0 \le j \le k'$, be the subset as given in Corollary 3.4, $c'_j = |W_j|$, and $C' = [c'_0, \ldots, c'_{k'}]$. Then there is a list $L' = [(a'_1,b'_1), \ldots, (a'_r,b'_r)]$ satisfying condition* **C3** *and a mapping from $C'_{>1}$ to L' satisfying condition* **C4**, *such that for all j, if $c'_j > 1$ is mapped to (a'_i,b'_i), then the function $\phi_{a'_i,b'_i,c'_j(c'_j-1)}$ is injective from W_j.*

Therefore, for the pair (a',b') and the list $C' = [c'_0, \ldots, c'_{k'}]$ in Corollary 3.4, which satisfy conditions **C1** and **C2**, respectively, and for the list $L' = [(a'_1,b'_1), \ldots, (a'_r,b'_r)]$ and the mapping from $C'_{>1}$ to L' in Corollary 3.6, which satisfy conditions **C3** and **C4**, respectively, each function $\phi_{a'_i,b'_i,c'_j(c'_j-1)}$ is injective from the subset W_j to $Z_{c'_j(c'_j-1)}$ for all j. For each j, let W'_j be the image of W_j under the function $\phi_{a'_i,b'_i,c'_j(c'_j-1)}$, then $W'_j \subseteq Z_{c'_j(c'_j-1)}$ and $|W'_j| = c'_j$. Since $\mathcal{F}_{c'_j(c'_j-1)}$ is a $(c'_j(c'_j-1), c'_j)$-family of perfect hashing functions, one $F_{c'_j}$ of the c'_j-colorings in $\mathcal{F}_{c'_j(c'_j-1)}$ is injective from W'_j. According to the algorithm **Coloring**, when this

c'_j-coloring $F_{c'_j}$ is used for the algorithm, the c'_j elements in W_j are colored with distinct colors. Running this for all j, we conclude that there is a list $[F_{c'_0}, F_{c'_1}, \ldots, F_{c'_{k'}}]$, where $F_{c'_j}$ is a c'_j-coloring in the $(c'_j(c'_j-1), c'_j)$-family $\mathcal{F}_{c'_j(c'_j-1)}$ of perfect hashing functions, satisfying condition **C5** such that all elements in the subset W are colored with distinct colors.

Summarizing the above discussion, we conclude

THEOREM 3.3. *For each subset W of k elements in Z_n, there is a combination of parameters satisfying conditions* **C1**–**C5** *on which the algorithm* **Coloring** *produces a k-coloring for Z_n that is injective from W.*

Combining Theorem 3.3 with Theorem 3.2 and by running the algorithm **Coloring** on all possible combinations of parameters satisfying conditions **C1**–**C5**, we obtain an (n,k)-family of perfect hashing functions of size $O(6.383^k k^{4\log k-4})$. For each k-coloring, the running time of the algorithm **Coloring** is $O(k^2)$. Recall that we have let $n=k^2$, we get

THEOREM 3.4. *For any integer k divisible by 4, a (k^2,k)-family of perfect hashing functions of size $O(6.383^k k^{4\log k-4})$ can be constructed in time $O(6.383^k k^{4\log k-2})$.*

Using Theorem 3.1, we can easily extend Theorem 3.4 to general values of n.

THEOREM 3.5. *For any integer n, and any integer k divisible by 4, an (n,k)-family of perfect hashing functions of size $O(6.383^k k^{O(\log k)}\log^2 n)$ can be constructed in time $O(6.383^k k^{O(\log k)} n\log^2 n)$.*

To extend Theorem 3.5 to general values of k, suppose that $k=4k'-h$, where $1\le h\le 3$. We first construct an $(n,4k')$-family $\mathcal{F}'$ of perfect hashing functions of size

$$\begin{aligned} & O(6.383^{4k'}(4k')^{O(\log 4k')}\log^2 n) \\ = \;& O(6.383^k k^{O(\log k)}\log^2 n). \end{aligned}$$

Now for each $(4k')$-coloring F in $\mathcal{F}'$, we construct $\binom{4k'}{h}=O(k^3)$ k-colorings for Z_n by selecting every subset of h colors in F and replacing them arbitrarily by the remaining $k=4k'-h$ colors. This gives a collection $\mathcal{F}$ of $O(6.383^k k^{O(\log k)}\log^2 n)=O(6.4^k\log^2 n)$ k-colorings for Z_n. To see that this is an (n,k)-family of perfect hashing functions, let W be any subset of k elements in Z_n. Let W' be a subset of $4k'$ elements in Z_n obtained from W by adding arbitrarily h elements. Since $\mathcal{F}'$ is an $(n,4k')$-family of perfect hashing functions, there is a $(4k')$-coloring F' in $\mathcal{F}'$ that is injective from W'. In particular, this $(4k')$-coloring F' is also injective from W. Now the k-coloring F in $\mathcal{F}$ obtained from F' by removing the other h colors is injective from W.

THEOREM 3.6. *For any integers n and k, where $n\ge k$, an (n,k)-family of perfect hashing functions of size $O(6.4^k\log^2 n)$ can be constructed in time $O(6.4^k n\log^2 n)$.*

4 An improved 3-D MATCHING algorithm

The method of perfect hashing function families seems to provide a fairly general technique for solving NP-hard problems, especially for those that are concerned with finding a small number of proper elements in a large set. In this section, we illustrate this method by presenting a further improved deterministic algorithm for the 3-D MATCHING problem.

Let the universal triple set be $U=X\times Y\times Z$, where X, Y, and Z are three disjoint symbol sets. The symbols in the sets X, Y, and Z will be called the symbols in *column-1*, *column-2*, and *column-3*, respectively. A *k-matching* is a set of k triples in which no two triples share a common symbol. The 3-D MATCHING problem is for a given set S of triples and an integer k, to determine whether S contains a k-matching.

The following theorem has been proved in [14].

THEOREM 4.1. *Let S be a triple set that contains a k-matching M_k, and let f be a g-coloring on the symbols in column-2 and column-3 in S such that no two symbols in M_k are colored with the same color. Then there is an algorithm that runs in time $O(\sum_{i=0}^{k}\binom{g}{2i}n)$ and returns a k-matching in S.*

Therefore, in order to construct a k-matching M_k in the triple set S, we only need to construct a g-coloring on the symbols in column-2 and column-3 in S so that no two symbols in column-2 and column-3 in M_k are colored with the same color. A straightforward way is to construct an $(m,2k)$-family of perfect hashing functions, where m is the total number of symbols in column-2 and column-3 in S. However, we can do even better. We start with the following theorem.

THEOREM 4.2. *Let S be a set of n triples and let M_{k-1} be a $(k-1)$-matching in S. If S contains a k-matching, then there is a k-matching M_k in S such that each triple in M_{k-1} contains at least two symbols in M_k.*

Therefore, if the $(k-1)$-matching M_{k-1} is given, then we are missing at most $3k-2(k-1)=k+2$ symbols in the k-matching M_k. In particular, we can assume without loss of generality that we are missing at most $2(k+2)/3$ column-2 and column-3 symbols in

M_k (in fact, we can rotate the columns and try all three possible cases). For this, we construct an (m', k')-family $\mathcal{F}$ of perfect hashing functions on the column-2 and column-3 symbols in $S - M_{k-1}$, where m' is the total number of column-2 and column-3 symbols in $S - M_{k-1}$, and $k' = 2(k+2)/3$. The size of the family $\mathcal{F}$ is bounded by $O(6.4^{k'} \log^2 m') = O(3.45^k \log^2 n)$. By the definition, one of the k'-colorings in the family $\mathcal{F}$ colors all column-2 and column-3 symbols in $M_k - M_{k-1}$ with different colors. Note that if we regard each column-2 and column-3 symbol in M_{k-1} as a color, then the $2(k-1)$ column-2 and column-3 symbols in M_{k-1} plus each k'-coloring in $\mathcal{F}$ constitutes a $((8k-2)/3)$-coloring for the column-2 and column-3 symbols in S. Therefore, the $2(k-1)$ column-2 and column-3 symbols in M_{k-1} and the family $\mathcal{F}$ make a family $\mathcal{F}'$ of $O(3.45^k \log^2 n)$ $((8k-2)/3)$-colorings in which at least one colors all column-2 and column-3 symbols in M_k with different colors.

Now it is clear how our algorithm proceeds. After constructing the family $\mathcal{F}'$ of $O(3.45^k \log^2 n)$ $((8k-2)/3)$-colorings, which can be done in time $O(3.45^k n \log^2 n)$, we color the column-2 and column-3 symbols in S using each coloring in $\mathcal{F}'$, and apply the algorithm in Theorem 4.1 to the colored instance. According to the theorem, the algorithm runs in time $O(2^{8k/3} n)$ for each coloring in $\mathcal{F}'$. Moreover, if the set S contains a k-matching, then by Theorem 4.1, one of the coloring in $\mathcal{F}'$ will make the algorithm in Theorem 4.1 to return a k-matching in S. The total running time of this process is bounded by $O(3.45^k \log^2 n \cdot 2^{8k/3} n) = O(21.91^k n \log^2 n) = O(2.80^{3k} n \log^2 n)$.

The only thing that remains is how we can construct the $(k-1)$-matching M_{k-1}. For this, we can start with a 1-matching to construct a 2-matching, and then from the 2-matching to construct a 3-matching, and so on. This iteration adds another factor of k to the running time. The following theorem summarizes our discussion above, which further improves Theorem 2.4.

THEOREM 4.3. *There is a deterministic algorithm of running time $O(2.80^{3k} kn \log^2 n)$ that solves the* 3-D MATCHING *problem.*

References

[1] N. Alon, O. Goldreich, J. Håstad, and R. Peralta, *Simple constructions of almost k-wise independent random variables*, Journal of Random Structures and Algorithms, 3 (1992), pp. 289–304.

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

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

[4] J. L. Carter, and M. N. Wegman, *Universal classes of hash functions*, Journal of Computer and System Sciences, 18 (1979), pp. 143–154.

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

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

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

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

[9] G. H. Hardy, and E. M. Wright, *An Introduction to the Theory of Numbers*, 5th ed., Oxford University Press, (1979).

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

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

[12] J. Kneis, D. Mölle, S. Richter, and P. Rossmanith, *Divide-and-color*, Lecture Notes in Computer Science, (WG 2006), to appear.

[13] I. Koutis, *A faster parameterized algorithm for set packing*, Information Processing Letters, 94 (2005), pp. 7–9.

[14] Y. Liu, S. Lu, J. Chen, and S.-H. Sze, *Greedy localization and color-coding: improved matching and packing algorithms*, Lecture Notes in Computer Science, 4169 (IWPEC 2006), pp. 84–95.

[15] B. Monien, *How to find long paths efficiently*, Annals of Discrete Mathematics, 25 (1985), pp. 239–254.

[16] M. Naor, L. J. Schulman, A. Srinivasan, *Splitters and near-optimal derandomization*, In Proceedings of the 39th IEEE Annual Symposium on Foundatins of Computer Science, (FOCS 1995), pp. 182–190.

[17] A. Nilli, *Perfect hashing and probability*, Probability and Computing, 3 (1994), pp. 407–409.

[18] C. Papadimitriou, and M. Yannakakis, *On limited nondeterminism and the complexity of the V-C dimension*, Journal of Computer and System Sciences, 53 (1996), pp. 161–170.

[19] J. P. Schmidt, and A. Siegel, *The spatial complexity of oblivious k-probe hash functions*, SIAM Journal on Computing, 19 (1990), pp. 775–786.

[20] J. Scott, T. Ideker, R. M. Karp, and R. Sharan, *Efficient algorithms for detecting signaling pathways in protein interaction networks*, Journal of Computational Biology, 13 (2006), pp. 133–144.