RWTH Aachen

Department of Computer Science
Technical Report

Derandomizing Non-uniform Color-Coding I

Joachim Kneis, Alexander Langer, Peter Rossmanith

ISSN 0935–3232 · Aachener Informatik Berichte · AIB-2009-07

RWTH Aachen · Department of Computer Science · March 2009

Derandomizing Non-uniform Color-Coding I*

Joachim Kneis, Alexander Langer, Peter Rossmanith

Theoretical Computer Science Group
RWTH Aachen University, Germany
Email: {kneis,langer,rossmani}@cs.rwth-aachen.de

Abstract. Color-coding, as introduced by Alon, Yuster, and Zwick, is a well-known tool for algorithm design and can often be efficiently derandomized using universal hash functions. In the special case of only two colors, one can use (n, k)-universal sets for the derandomization. In this paper, we introduce (n, k, l)-universal sets that are typically smaller and can be constructed faster. Nevertheless, for some problems they are still sufficient for derandomization and faster deterministic algorithms can be obtained. This particularly works well when the color-coding does not use a uniform probability distribution. To exemplify the concept, we present an algorithm for the Unique Coverage problem introduced by Demaine, Feige, Hajiaghayi, and Salavatipour. The example also shows how to extend the concept to multiple colors.

1 Introduction

The principle of randomization is well-established in the field of algorithm design. Due to the sometimes non-intuitive, even surprising laws of probability theory (cf., the *birthday paradoxon*), randomized algorithms often can find correct solutions faster than any known deterministic algorithm.

Consider, for example, the popular 3-SAT problem, which can be solved in time bounded by $O^*(1.324^n)$ using a randomized algorithm with constant error probability [10, 17], but only in time bounded by $O^*(1.473^n)$ when a purely deterministic approach is used [6, 2].

Even worse, it can be shown that there are problems, for which *every* deterministic approach has a certain worst-case lower bound, while a randomized algorithm typically runs much faster. Suppose, for example, that there are 100 boxes and exactly half of them contain a present. How long does it take to find one? Every deterministic algorithm can be forced to look into at least 51 boxes before finding a present. However, a randomized algorithm repeatedly choosing an arbitrary box with uniform probability 1/100 until finding a present is expected to guess only two times. Here, the worst- and average-case coincide, which explains the large gap between deterministic and randomized bounds.

Similarly, there are examples that are not so naïve, yet surprising: Let's say, we are are given an array of n integers, and we know that either the array is sorted, or it is not sorted and you have to delete at least $n/10$ elements to obtain a sorted array. The goal is to design an algorithm that decides which of the two cases holds. Intuitively, one could argue you always have to compare all pairs of subsequent elements, and in fact deterministically $\Omega(n)$ elements have to be examined. Surprisingly, there is a randomized algorithm that runs in $O(\log n)$ time [8].

* Supported by the DFG under grant RO 927/8

Randomized algorithms are algorithms that have access to true random bits. They can be classified into *Monte Carlo* and *Las Vegas* algorithms: Monte Carlo algorithms have a fixed running time, but the outcome has to be correct with a probability of at least, say, 2/3. Las Vegas algorithms always return the correct result, but their running time depends on the random bits. In this paper, we consider Monte Carlo algorithms.

Often you can turn a randomized algorithm into a deterministic one by techniques commonly referred to as *derandomization*. One possibility is to repeatedly feed the algorithm several bit strings of length n instead of n random bits. Usually, however, a randomized algorithm requires random bits that are independent, i.e., for n random bits, every combination occurs with equal probability 2^{-n}. One possibility to derandomize such an algorithm is therefore to run it 2^n times with every possible bit string of length n, which is usually too slow to be of practical interest.

Sometimes, though, algorithms do not require n independent random bits. Sometimes, it suffices if only every subset of $k < n$ bits is independent. A set of n-bit vectors with this property is called (n, k)-*independent*. Such randomized algorithms that only require (n, k)-independent bits can often be derandomized with the help of (n, k)-*universal sets*.

Definition 1. *Let n and k be integers. An (n, k)-universal set is a set Ω of bit strings $b = b_0 \dots b_{n-1} \in \{0,1\}^n$, such that for all distinct positions $i_1, \dots, i_k \in Z_n$ and all patterns $x = x_1 \dots x_k \in \{0,1\}^k$ there is some $b = b_0 \dots b_{n-1} \in \Omega$ with $b_{i_j} = x_j$ for all $1 \leq j \leq k$.*

Naor, Schulman, and Srinivasan [15] showed how to construct (n, k)-universal sets of size $2^{k+O(\log^2 k)} \log n$ in linear time. This is nearly optimal, because there is a lower bound of $\Omega(2^k \log n)$ on the size of (n, k)-universal sets [11].

Algorithms that can often be derandomized using (n, k)-universal sets are those that use the so called *color-coding* technique [1], the *random separation* technique [3], and those that use the *randomized divide-and-conquer* (also called *divide-and-color*) [4, 5, 12] approach. In Section 2.1, we illustrate the random separation technique in further detail on the problem Exact Partial Vertex Cover: Given a graph and two integers k and t, is there a set C of k nodes adjacent to exactly t edges? In short, the algorithm randomly partitions the nodes into two sets colored 0 and 1, and then uses dynamic programming to find a solution C that is colored with color 1 and whose neighborhood is colored with color 0. This coloring step succeeds with a probability of at least 2^{-k-t} when uniform probabilities for the two colors are used.[1] Furthermore, (n, k)-universal sets can be used to obtain a deterministic algorithm with a running time of $2^{k+t+O(\log^2(k+t))} poly(n)$.

Sometimes, however, it is not optimal to choose the colors with uniform probabilities. This is, for example, the case, when much fewer elements shall receive the color 0 than shall receive the color 1. Assume, for instance, that an

[1] Note that a randomized algorithm with running time $t(n)$ and success probability 2^{-x} can easily be turned into a randomized algorithm with constant success probability arbitrarily close to one and a running time of $O(2^x t(n))$. This standard technique is called *probability amplification* (see, e.g., [14]). Whenever we compare randomized algorithms with deterministic algorithms, we mean the randomized algorithm with constant success probability.

instance of Exact Partial Vertex Cover is such that $t = k^2$, and therefore k^2 nodes shall be colored with 0, but only k nodes with 1. Then the randomized algorithm's success probability is only 2^{-k^2-k} when using uniform probabilities, but can be improved to

$$(1/k)^k(1-1/k)^{k^2} \sim e^{-k\ln k-k+1/2}$$

by using probabilities $1/k$ for the color 1 and $1-1/k$ for the color 0. See Section 2.2 for details. If we now simply use (n, k^2+k)-universal sets for the derandomization, we still only obtain a deterministic algorithm with running time of $2^{k^2+k+O(\log k)}poly(n)$, which is much slower than the corresponding randomized algorithm.

Please note, however, that here we do not need the whole power of (n, k^2+k)-universal sets: We do not need to find all possible subpatterns, but only those having k ones. To capture this concept, we introduce (n, k, l)-*universal sets*[2]:

Definition 2. *Let $n \geq k \geq l$ be integers. An (n, k, l)-universal set is a set Ω of bit strings $b = b_0 \dots b_{n-1} \in \{0,1\}^n$, such that for all distinct positions $i_1, \dots, i_k \in Z_n$ and all patterns $x = x_1 \dots x_k \in \{0,1\}^k$ with Hamming weight at most l, there is some $b = b_0 \dots b_{n-1} \in \Omega$ with $b_{i_j} = x_j$ for all $1 \leq j \leq k$.*

Using (n, k^2+k, k)-universal sets, we can get a better deterministic time bound if we are able to construct (n, k^2+k, k)-universal sets much faster than (n, k^2+k)-universal sets. The main result of this paper are (n, k, l)-universal sets of size $2nk^{2l}$ that can be constructed in time $O(n^2k^{2l+2})$.

1.1 Outline

In this paper, we present a construction of (n, k, l)-universal sets, which is self-contained, very simple, and can easily be implemented, but has two drawbacks:

1. The size of the sets is not the smallest possible.
2. The method works well for $l \ll k$, but is not optimal for $l = \Theta(k)$, in particular for $l = k/2$.

These issues will be addressed in the second part, where we will present a more complicated technique, which allows for better constants and is useful for all combinations of k and l.

This paper is organized as follows: In Section 2.1 we introduce the concept of *random separation* with uniform probabilities using the example of the Exact Partial Vertex Cover problem. In Section 2.2, we show how the running time can be improved when certain conditions are met and motivate the introduction of (n, k, l)-universal sets. Our construction of (n, k, l)-universal sets is presented in Section 3. Another application is shown in Section 4, which contains the currently fastest parameterized algorithm for the Unique Coverage problem introduced by Demaine, Feige, Hajiaghayi, and Salavatipour [7].

[2] Please do not confuse those with (n, k, l)-splitters [15], which are k-universal hash functions on l colors, i.e., a $(n, k, 2)$-splitter is also an (n, k)-universal set.