A Reliable Randomized Algorithm for the Closest-Pair Problem

Martin Dietzfelbinger *

Fachbereich Informatik
Universität Dortmund
D-44221 Dortmund, Germany

Torben Hagerup †

Max-Planck-Institut für Informatik
Im Stadtwald
D-66123 Saarbrücken, Germany

Jyrki Katajainen ‡

Datalogisk Institut
Københavns Universitet
Universitetsparken 1
DK-2100 København Ø, Denmark

Martti Penttonen §

Tietojenkäsittelytieteen laitos
Joensuun yliopisto
PL 111
FIN-80101 Joensuu, Finland

*Partially supported by DFG grant Me 872/1-4.

†Partially supported by the ESPRIT Basic Research Actions Program of the EC under contract No. 7141 (project ALCOM II).

‡Partially supported by the Academy of Finland under contract No. 1021129 (project "Efficient Data Structures and Algorithms").

§Partially supported by the Academy of Finland.

Running head:
A RELIABLE RANDOMIZED ALGORITHM FOR CLOSEST PAIRS

For correspondence use:

Jyrki Katajainen

Datalogisk Institut
Københavns Universitet
Universitetsparken 1
DK-2100 København Ø, Denmark

telephone: +45 35 32 14 00
telefax: +45 35 32 14 01
e-mail: jyrki@diku.dk

Abstract

The following two computational problems are studied:

Duplicate grouping: Assume that n items are given, each of which is labeled by an integer key from the set $\{0, \ldots, U-1\}$. Store the items in an array of size n such that items with the same key occupy a contiguous segment of the array.

Closest pair: Assume that a multiset of n points in the d-dimensional Euclidean space is given, where $d \geq 1$ is a fixed integer. Each point is represented as a d-tuple of integers in the range $\{0, \ldots, U-1\}$ (or of arbitrary real numbers). Find a closest pair, i. e., a pair of points whose distance is minimal over all such pairs.

In 1976 Rabin described a randomized algorithm for the closest-pair problem that takes linear expected time. As a subroutine, he used a hashing procedure whose implementation was left open. Only years later randomized hashing schemes suitable for filling this gap were developed.

In this paper, we return to Rabin's classic algorithm in order to provide a fully detailed description and analysis, thereby also extending and strengthening his result. As a preliminary step, we study randomized algorithms for the duplicate-grouping problem. In the course of solving the duplicate-grouping problem, we describe a new universal class of hash functions of independent interest.

It is shown that both of the problems above can be solved by randomized algorithms that use $O(n)$ space and finish in $O(n)$ time with probability tending to 1 as n grows to infinity. The model of computation is a unit-cost RAM capable of generating random numbers and of performing arithmetic operations from the set $\{+, -, *, \text{DIV}, \text{LOG}_2, \text{EXP}_2\}$, where DIV denotes integer division and LOG_2 and EXP_2 are the mappings from $\mathbb{N}$ to $\mathbb{N} \cup \{0\}$ with $\text{LOG}_2(m) = \lfloor \log_2 m \rfloor$ and $\text{EXP}_2(m) = 2^m$, for all $m \in \mathbb{N}$. If the operations LOG_2 and EXP_2 are not available, the running time of the algorithms increases by an additive term of $O(\log \log U)$. All numbers manipulated by the algorithms consist of $O(\log n + \log U)$ bits.

The algorithms for both of the problems exceed the time bound $O(n)$ or $O(n + \log \log U)$ with probability $2^{-n^{\Omega(1)}}$. Variants of the algorithms are also given that use only $O(\log n + \log U)$ random bits and have probability $O(n^{-\alpha})$ of exceeding the time bounds, where $\alpha \geq 1$ is a constant that can be chosen arbitrarily.

The algorithm for the closest-pair problem also works if the coordinates of the points are arbitrary real numbers, provided that the RAM is able to perform arithmetic operations from $\{+, -, *, \text{DIV}\}$ on real numbers, where $a \,\text{DIV}\, b$ now means $\lfloor a/b \rfloor$. In this case, the running time is $O(n)$ with LOG_2 and EXP_2 and $O(n + \log \log(\delta_{\max}/\delta_{\min}))$ without them, where $\delta_{\max}$ is the maximum and $\delta_{\min}$ is the minimum distance between any two distinct input points.

LIST OF SYMBOLS

∞	infinity symbol
$\sum$	summation symbol
Ω	cap. omega
$\mathcal{H}$	calligraphic aych
$\mathbb{R}$	the set of reals
$\mathbb{N}$	the set of natural numbers
$\mathbb{Z}$	the set of integers

A	cap. ay	a	lower ay	α	alpha
B	cap. bee	b	lower bee	β	beta
		c	lower cee		
D	cap. dee	d	lower dee	δ	delta
				ε	varepsilon
G	cap. gee			γ	gamma
		h	lower aych		
		i	lower eye		
		j	lower jay		
		k	lower kay		
L	cap. ell	ℓ	lower ell		
		m	lower em	μ	mu
N	cap. en	n	lower en		
		o	lower oh		
P	cap. pee	p	lower pee	π	pi
R	cap. are	r	lower are		
S	cap. ess	s	lower ess		
		t	lower tee		
U	cap. you				
X	cap. eks	x	lower eks		
		y	lower why		
		z	lower zee		
0	zero				
1	one				

The LaTeX source of this paper is available from the authors as a typesetting aid.

1 Introduction

The *closest-pair problem* is often introduced as the first nontrivial proximity problem in computational geometry—see, e. g., [26]. In this problem we are given a collection of n points in d-dimensional space, where $d \geq 1$ is a fixed integer, and a metric specifying the distance between points. The task is to find a pair of points whose distance is minimal. We assume that each point is represented as a d-tuple of real numbers, or of integers in a fixed range, and that the distance measure is the standard Euclidean metric.

In his seminal paper on randomized algorithms, Rabin [27] proposed an algorithm for solving the closest-pair problem. The key idea of the algorithm is to determine the minimal distance δ_0 within a random sample of points. When the points are grouped according to a grid with resolution δ_0, the points of a closest pair fall in the same cell or in neighboring cells. This considerably decreases the number of possible closest-pair candidates from the total of $n(n-1)/2$. Rabin proved that with a suitable sample size the total number of distance calculations performed will be of order n with overwhelming probability.

A question that was not solved satisfactorily by Rabin is how the points are grouped according to a δ_0-grid. Rabin suggested that this could be implemented by dividing the coordinates of the points by δ_0, truncating the quotients to integers, and hashing the resulting integer d-tuples. Fortune and Hopcroft [15], in their more detailed examination of Rabin's algorithm, assumed the existence of a special operation $findbucket(\delta_0, p)$, which returns an index of the cell into which the point p falls in some fixed δ_0-grid. The indices are integers in the range $\{1, \ldots, n\}$, and distinct cells have distinct indices.

On a real RAM (for the definition see [26]), where the generation of random numbers, comparisons, arithmetic operations from $\{+, -, *, /, \sqrt{\ }\}$, and $findbucket$ require unit time, Rabin's random-sampling algorithm runs in $O(n)$ expected time [27]. (Under the same assumptions the closest-pair problem can even be solved in $O(n \log\log n)$ time in the worst case, as demonstrated by Fortune and Hopcroft [15].) We next introduce terminology that allows us to characterize the performance of Rabin's algorithm more closely. Every execution of a randomized algorithm *succeeds* or *fails*. The meaning of "failure" depends on the context, but an execution typically fails if it produces an incorrect result or does not finish in time. We say that a randomized algorithm is *exponentially reliable* if, on inputs of size n, its failure probability is bounded by 2^{-n^ε} for some fixed $\varepsilon > 0$. Rabin's algorithm is exponentially reliable. Correspondingly, an algorithm is *polynomially reliable* if, for every fixed $\alpha > 0$, its failure probability on inputs of size n is at most $n^{-\alpha}$. In the latter case, we allow the notion of success to depend on α; an example is the expression "runs in linear time", where the constant implicit in the term "linear" may (and usually will) be a function of α.

Recently, two other simple closest-pair algorithms were proposed by Golin et al. [16] and Khuller and Matias [19]; both algorithms offer linear expected running time. Faced with the need for an implementation of the $findbucket$ operation, these papers employ randomized hashing schemes that had been developed in the

meantime [8, 14]. Golin et al. present a variant of their algorithm that is polynomially reliable but has running time $O(n \log n / \log\log n)$ (this variant utilizes the polynomially reliable hashing scheme of [13]).

The time bounds above should be contrasted with the fact that in the algebraic computation-tree model (where the available operations are comparisons and arithmetic operations from $\{+, -, *, /, \sqrt{\ }\}$, but where indirect addressing is not modeled), $\Theta(n \log n)$ is known to be the complexity of the closest-pair problem. Algorithms proving the upper bound were provided by, for example, Bentley and Shamos [7] and Schwarz et al. [30]. The lower bound follows from the corresponding lower bound derived for the element-distinctness problem by Ben-Or [6]. The $\Omega(n \log n)$ lower bound is valid even if the coordinates of the points are integers [32] or if the sequence of points forms a simple polygon [1].

The present paper centers on two issues: First, we completely describe an implementation of Rabin's algorithm, including all the details of the hashing subroutines, and show that it guarantees linear running time together with exponential reliability. Second, we modify Rabin's algorithm so that only very few random bits are needed, but still a polynomial reliability is maintained.[1]

As a preliminary step, we address the question of how the grouping of points can be implemented when only $O(n)$ space is available and the strong $findbucket$ operation does not belong to the repertoire of available operations. An important building block in the algorithm is an efficient solution to the *duplicate-grouping problem* (sometimes called the *semisorting problem*), which can be formulated as follows: Given a set of n items, each of which is labeled by an integer key from $\{0, \ldots, U-1\}$, store the items in an array A of size n so that entries with the same key occupy a contiguous segment of the array, i. e., if $1 \le i < j \le n$ and $A[i]$ and $A[j]$ have the same key, then $A[k]$ has the same key for all k with $i \le k \le j$. Note that full sorting is not necessary, since no order is prescribed for items with different keys. In a slight generalization, we consider the duplicate-grouping problem also for keys that are d-tuples of elements from the set $\{0, \ldots, U-1\}$, for some integer $d \ge 1$.

We provide two randomized algorithms for dealing with the duplicate-grouping problem. The first one is very simple; it combines universal hashing [8] with (a variant of) radix sort [2, pp. 77 ff.] and runs in linear time with polynomial reliability. The second method employs the exponentially reliable hashing scheme of [4]; it results in a duplicate-grouping algorithm that runs in linear time with exponential reliability. Assuming that U is a power of 2 given as part of the input, these algorithms use only arithmetic operations from $\{+, -, *, \text{DIV}\}$. If U is not known, we have to spend $O(\log\log U)$ preprocessing time on computing a power of 2 greater than the largest input number. That is, the running time is linear if $U = 2^{2^{O(n)}}$. Alternatively, we get linear running time if we accept LOG_2 and EXP_2 among the unit-time operations. It is essential to note that our

[1] In the algorithms of this paper randomization occurs in computational steps like "pick a random number in the range $\{0, \ldots, r-1\}$ (according to the uniform distribution)". Informally we say that such a step "uses $\lceil \log_2 r \rceil$ random bits".

algorithms for duplicate grouping are *conservative* in the sense of [20], i. e., all numbers manipulated during the computation have $O(\log n + \log U)$ bits.

Technically as an ingredient of the duplicate-grouping algorithms, we introduce a new universal class of hash functions—more precisely, we prove that the class of multiplicative hash functions [21, pp. 509–512] is universal in the sense of [8]. The functions in this class can be evaluated very efficiently, using only multiplications and shifts of binary representations. These properties of multiplicative hashing are crucial to its use in the signature-sort algorithm of [3].

On the basis of the duplicate-grouping algorithms we give a rigorous analysis of several variants of Rabin's algorithm, including all the details concerning the hashing procedures. For the core of the analysis, we use an approach completely different from that of Rabin, which enables us to show that the algorithm can also be run with very few random bits. Further, the analysis of the algorithm is extended to cover the case of repeated input points. (Rabin's analysis was based on the assumption that all input points are distinct.) The result returned by the algorithm is always correct; with high probability, the running time is bounded as follows: On a real RAM with arithmetic operations from $\{+, -, *, \text{DIV}, \text{LOG}_2, \text{EXP}_2\}$, the closest-pair problem is solved in $O(n)$ time, and with operations from $\{+, -, *, \text{DIV}\}$ it is solved in $O(n + \log\log(\delta_{\max}/\delta_{\min}))$ time, where $\delta_{\max}$ is the maximum and $\delta_{\min}$ is the minimum distance between distinct input points (here $a \,\text{DIV}\, b$ means $\lfloor a/b \rfloor$, for arbitrary positive real numbers a and b). For points with integer coordinates in the range $\{0, \ldots, U-1\}$ the latter running time can be estimated by $O(n + \log\log U)$. For integer data, the algorithms are again conservative.

The rest of the paper is organized as follows. In Section 2, the algorithms for the duplicate-grouping problem are presented. The randomized algorithms are based on the universal class of multiplicative hash functions. The randomized closest-pair algorithm is described in Section 3 and analyzed in Section 4. The last section contains some concluding remarks and comments on experimental results. Technical proofs regarding the problem of generating primes and probability estimates are given in the two parts of an appendix.

2 Duplicate grouping

In this section we present two simple deterministic algorithms and two randomized algorithms for solving the duplicate-grouping problem. As a technical tool, we describe and analyze a new, simple universal class of hash functions. Moreover, a method for generating numbers that are prime with high probability is provided.

An algorithm is said to rearrange a given sequence of items, each with a distinguished key, *stably* if items with identical keys appear in the input in the same order as in the output. In order to simplify notation in the following, we will ignore all components of the items excepting the keys; in other words, we will consider the problem of duplicate grouping for inputs that are multisets of integers or multisets of tuples of integers. It will be obvious that the algorithms to be

presented can be extended to solve the more general duplicate-grouping problem in which additional data is associated with the keys.

2.1 Deterministic duplicate grouping

We start with a trivial observation: Sorting the keys certainly solves the duplicate-grouping problem. In our context, where linear running time is essential, variants of radix sort [2, pp. 77 ff.] are particularly relevant.

Fact 2.1 [2, p. 79] *The sorting problem (and hence the duplicate-grouping problem) for a multiset of n integers from $\{0, \ldots, n^\beta - 1\}$ can be solved stably in $O(\beta n)$ time and $O(n)$ space, for any integer $\beta \geq 1$. In particular, if β is a fixed constant, both time and space are linear.*

Remark 2.2 Recall that radix sort uses the digits of the n-ary representation of the keys being sorted. For justifying the space bound $O(n)$ (instead of the more natural $O(\beta n)$), observe that it is not necessary to generate and store the full n-ary representation of the integers being sorted, but that it suffices to generate a digit when it is needed. Since the modulo operation can be expressed in terms of DIV, $*$, and $-$, generating such a digit needs constant time on a unit-cost RAM with operations from $\{+, -, *, \text{DIV}\}$.

If space is not an issue, there is a simple algorithm for duplicate grouping that runs in linear time and does not sort. It works similarly to one phase of radix sort, but avoids scanning the range of all possible key values in a characteristic way.

Lemma 2.3 *The duplicate-grouping problem for a multiset of n integers from $\{0, \ldots, U-1\}$ can be solved stably by a deterministic algorithm in time $O(n)$ and space $O(n + U)$.*

Proof. For definiteness, assume that the input is stored in an array S of size n. Let L be an auxiliary array of size U, which is indexed from 0 to $U - 1$ and whose possible entries are headers of lists (this array need not be initialized). The array S is scanned three times from index 1 to index n. During the first scan, for $i = 1, \ldots, n$, the entry $L[S[i]]$ is initialized to point to an empty list. During the second scan, the element $S[i]$ is inserted at the end of the list with header $L[S[i]]$. During the third scan, the groups are output as follows: for $i = 1, \ldots, n$, if the list with header $L[S[i]]$ is nonempty, it is written to consecutive positions of the output array and $L[S[i]]$ is made to point to an empty list again. Clearly, this algorithm runs in linear time and groups the integers stably. ∎

In our context, the algorithms for the duplicate-grouping problem considered so far are not sufficient since there is no bound on the sizes of the integers that may appear in our geometric application. The radix-sort algorithm might be slow and the naive duplicate-grouping algorithm might waste space. Both time and space efficiency can be achieved by compressing the numbers by means of hashing, as will be demonstrated in the following.

2.2 Multiplicative universal hashing

In order to prepare for the randomized duplicate-grouping algorithms, we describe a simple class of hash functions that is universal in the sense of Carter and Wegman [8]. Assume that $U \geq 2$ is a power of 2, say $U = 2^k$. For $\ell \in \{1, \ldots, k\}$, consider the class $\mathcal{H}_{k,\ell} = \{h_a \mid 0 < a < 2^k, \text{ and } a \text{ is odd}\}$ of hash functions from $\{0, \ldots, 2^k - 1\}$ to $\{0, \ldots, 2^\ell - 1\}$, where h_a is defined by

$$h_a(x) = (ax \mod 2^k) \operatorname{div} 2^{k-\ell}, \text{ for } 0 \leq x < 2^k.$$

The class $\mathcal{H}_{k,\ell}$ contains 2^{k-1} (distinct) hash functions. Since we assume that on the RAM model a random number can be generated in constant time, a function from $\mathcal{H}_{k,\ell}$ can be chosen at random in constant time, and functions from $\mathcal{H}_{k,\ell}$ can be evaluated in constant time on a RAM with arithmetic operations from $\{+, -, *, \text{DIV}\}$ (for this 2^k and 2^ℓ must be known, but not k or ℓ).

The most important property of the class $\mathcal{H}_{k,\ell}$ is expressed in the following lemma.

Lemma 2.4 *Let k and ℓ be integers with $1 \leq \ell \leq k$. If $x, y \in \{0, \ldots, 2^k - 1\}$ are distinct and $h_a \in \mathcal{H}_{k,\ell}$ is chosen at random, then*

$$\mathbf{Prob}\big(h_a(x) = h_a(y)\big) \leq \frac{1}{2^{\ell-1}}.$$

Proof. Fix distinct integers $x, y \in \{0, \ldots, 2^k - 1\}$ with $x > y$ and abbreviate $x - y$ by z. Let $A = \{a \mid 0 < a < 2^k \text{ and } a \text{ is odd}\}$. By the definition of h_a, every $a \in A$ with $h_a(x) = h_a(y)$ satisfies

$$|ax \bmod 2^k - ay \bmod 2^k| < 2^{k-\ell}.$$

Since $z \not\equiv 0 \pmod{2^k}$ and a is odd, we have $az \not\equiv 0 \pmod{2^k}$. Therefore all such a satisfy

$$az \bmod 2^k \in \{1, \ldots, 2^{k-\ell} - 1\} \cup \{2^k - 2^{k-\ell} + 1, \ldots, 2^k - 1\}. \tag{2.1}$$

In order to estimate the number of $a \in A$ that satisfy (2.1), we write $z = z'2^s$ with z' odd and $0 \leq s < k$. Since the odd numbers $1, 3, \ldots, 2^k - 1$ form a group with respect to multiplication modulo 2^k, the mapping

$$a \mapsto az' \bmod 2^k$$

is a permutation of A. Consequently, the mapping

$$a2^s \mapsto az'2^s \bmod 2^{k+s} = az \bmod 2^{k+s}$$

is a permutation of the set $\{a2^s \mid a \in A\}$. Thus, the number of $a \in A$ that satisfy (2.1) is the same as the number of $a \in A$ that satisfy

$$a2^s \bmod 2^k \in \{1, \ldots, 2^{k-\ell} - 1\} \cup \{2^k - 2^{k-\ell} + 1, \ldots, 2^k - 1\}. \tag{2.2}$$

Now, $a2^s \bmod 2^k$ is just the number whose binary representation is given by the $k-s$ least significant bits of a, followed by s zeroes. This easily yields the following. If $s \geq k - \ell$, no $a \in A$ satisfies (2.2). For smaller s, the number of $a \in A$ satisfying (2.2) is at most $2^{k-\ell}$. Hence the probability that a randomly chosen $a \in A$ satisfies (2.1) is at most $2^{k-\ell}/2^{k-1} = 1/2^{\ell-1}$. ∎

Remark 2.5 The lemma says that the class $\mathcal{H}_{k,\ell}$ of multiplicative hash functions is 2-universal in the sense of [24, p. 140] (this notion slightly generalizes that of [8]). As discussed in [21, p. 509] ("the multiplicative hashing scheme"), the functions in this class are particularly simple to evaluate, since the division and the modulo operation correspond to selecting a segment of the binary representation of the product ax, which can be done by means of shifts. Other universal classes use functions that involve division by prime numbers [8, 14], arithmetic in finite fields [8], matrix multiplication [8], or convolution of binary strings over the two-element field [22], i. e., operations that are more expensive than multiplications and shifts unless special hardware is available.

It is worth noting that the class $\mathcal{H}_{k,\ell}$ of multiplicative hash functions may be used to improve the efficiency of the static and dynamic perfect-hashing schemes described in [14] and [12], in place of the functions of the type $x \mapsto (ax \bmod p) \bmod m$, for a prime p, which are used in these papers, and which involve integer division. For an experimental evaluation of this approach, see [18]. In another interesting development, Raman [29] has shown that the so-called method of conditional probabilities can be used to obtain a function in $\mathcal{H}_{k,\ell}$ with desirable properties ("few collisions") in a deterministic manner (previously known deterministic methods for this purpose use exhaustive search in suitable probability spaces [14]); this allowed him to derive an efficient deterministic scheme for the construction of perfect hash functions.

The following is a well-known property of universal classes.

Lemma 2.6 *Let n, k and ℓ be positive integers with $\ell \le k$ and let S be a set of n integers in the range $\{0, \ldots, 2^k - 1\}$. Choose $h \in \mathcal{H}_{k,\ell}$ at random. Then*

$$\mathbf{Prob}(h \text{ is 1–1 on } S) \ge 1 - \frac{n^2}{2^\ell}.$$

Proof. By Lemma 2.4,

$$\mathbf{Prob}\Big(h(x) = h(y) \text{ for some } x, y \in S\Big) \le \binom{n}{2} \cdot \frac{1}{2^{\ell-1}} \le \frac{n^2}{2^\ell}.$$

■

2.3 Duplicate grouping via universal hashing

Having provided the universal class $\mathcal{H}_{k,\ell}$, we are now ready to describe our first randomized duplicate-grouping algorithm.

Theorem 2.7 *Let $U \ge 2$ be known and a power of 2 and let $\alpha \ge 1$ be an arbitrary integer. The duplicate-grouping problem for a multiset of n integers in the range $\{0, \ldots, U-1\}$ can be solved stably by a conservative randomized algorithm that needs $O(n)$ space and $O(\alpha n)$ time on a unit-cost RAM with arithmetic operations from $\{+, -, *, \text{DIV}\}$; the probability that the time bound is exceeded is bounded by $n^{-\alpha}$. The algorithm requires fewer than $\log_2 U$ random bits.*

Proof. Let S be the multiset of n integers from $\{0, \ldots, U-1\}$ to be grouped. Further, let $k = \log_2 U$ and $\ell = \lceil (\alpha + 2) \log_2 n \rceil$ and assume without loss of generality that $1 \leq \ell \leq k$. As a preparatory step, we compute 2^ℓ. The elements of S are then grouped as follows. First, a hash function h from $\mathcal{H}_{k,\ell}$ is chosen at random. Second, each element of S is mapped under h to the range $\{0, \ldots, 2^\ell - 1\}$. Third, the resulting pairs $(x, h(x))$, where $x \in S$, are sorted by radix sort (Fact 2.1) according to their second components. Fourth, it is checked whether all elements of S that have the same hash value are in fact equal. If this is the case, the third step has produced the correct result; if not, the whole input is sorted, e. g., with mergesort.

The computation of 2^ℓ is easily carried out in $O(\alpha \log n)$ time. The four steps of the algorithm proper require $O(1)$, $O(n)$, $O(\alpha n)$, and $O(n)$ time, respectively. Hence, the total running time is $O(\alpha n)$. The result of the third step is correct if h is 1–1 on the (distinct) elements of S, which happens with probability

$$\mathbf{Prob}(h \text{ is 1–1 on } S) \geq 1 - \frac{n^2}{2^\ell} \geq 1 - \frac{1}{n^\alpha}$$

by Lemma 2.6. In case the final check indicates that the outcome of the third step is incorrect, the call of mergesort produces a correct output in $O(n \log n)$ time, which does not impair the linear expected running time. The space requirements of the algorithm are dominated by those of the sorting subroutines, which need $O(n)$ space. Since both radix sort and mergesort rearrange the elements stably, duplicate grouping is performed stably. It is immediate that the algorithm is conservative and that the number of random bits needed is $k - 1 < \log_2 U$. ∎

2.4 Duplicate grouping via perfect hashing

We now show that there is another, asymptotically even more reliable, duplicate-grouping algorithm that also works in linear time and space. The algorithm is based on the randomized perfect-hashing scheme of Bast and Hagerup [4].

The *perfect-hashing problem* is the following: Given a multiset $S \subseteq \{0, \ldots, U-1\}$, for some universe size U, construct a function $h\colon S \to \{0, \ldots, c|S|\}$, for some constant c, so that h is 1–1 on (the distinct elements of) S. In [4] a parallel algorithm for the perfect-hashing problem is described; we need the following sequential version.

Fact 2.8 [4] *Assume that U is a known prime. Then the perfect-hashing problem for a multiset of n integers from $\{0, \ldots, U-1\}$ can be solved by a randomized algorithm that requires $O(n)$ space and runs in $O(n)$ time with probability $1 - 2^{-n^{\Omega(1)}}$. The hash function produced by the algorithm can be evaluated in constant time.*

In order to use this perfect-hashing scheme, we need to have a method for computing a prime larger than a given number m. In order to find such a prime, we again use a randomized algorithm. The simple idea is to combine a randomized

primality test (as described, e. g., in [10, pp. 839 ff.]) with random sampling. Such algorithms for generating a number that is probably prime are described or discussed in several papers, e. g., in [5], [11], and [23]. As we are interested in the situation where the running time is guaranteed and the failure probability is extremely small, we use a variant of the algorithms tailored to meet these requirements. The proof of the following lemma, which includes a description of the algorithm, can be found in Section A of the appendix.

Lemma 2.9 *There is a randomized algorithm that, for any given positive integers m and n with $2 \le m \le 2^{\lceil n^{1/4} \rceil}$, returns a number p with $m < p \le 2m$ such that the following holds: the running time is $O(n)$, and the probability that p is not prime is at most $2^{-n^{1/4}}$.*

Remark 2.10 The algorithm of Lemma 2.9 runs on a unit-cost RAM with operations from $\{+, -, *, \text{DIV}\}$. The storage space required is constant. Moreover, all numbers manipulated contain $O(\log m)$ bits.

Theorem 2.11 *Let $U \ge 2$ be known and a power of 2. The duplicate-grouping problem for a multiset of n integers in the range $\{0, \ldots, U-1\}$ can be solved stably by a conservative randomized algorithm that needs $O(n)$ space on a unit-cost RAM with arithmetic operations from $\{+, -, *, \text{DIV}\}$, so that the probability that more than $O(n)$ time is used is $2^{-n^{\Omega(1)}}$.*

Proof. Let S be the multiset of n integers from $\{0, \ldots, U-1\}$ to be grouped. Let us call U *large* if it is larger than $2^{\lceil n^{1/4} \rceil}$ and take $U' = \min\{U, 2^{\lceil n^{1/4} \rceil}\}$. We distinguish between two cases. If U is not large, i. e., $U = U'$, we first apply the method of Lemma 2.9 to find a prime p between U and $2U$. Then, the hash function from Fact 2.8 is applied to map the distinct elements of $S \subseteq \{0, \ldots, p-1\}$ to $\{0, \ldots, cn\}$, where c is a constant. Finally, the values obtained are grouped by one of the deterministic algorithms described in Section 2.1 (Fact 2.1 and Lemma 2.3 are equally suitable). In case U is large, we first "collapse the universe" by mapping the elements of $S \subseteq \{0, \ldots, U-1\}$ into the range $\{0, \ldots, U'-1\}$ by a randomly chosen multiplicative hash function, as described in Section 2.2. Then, using the "collapsed" keys, we proceed as above for a universe that is not large.

Let us now analyze the resource requirements of the algorithm. It is easy to check (conservatively) in $O(\min\{n^{1/4}, \log U\})$ time whether or not U is large. Lemma 2.9 shows how to find the required prime p in the range $\{U'+1, \ldots, 2U'\}$ in $O(n)$ time with error probability at most $2^{-n^{1/4}}$. In case U is large, we must choose a function h at random from $\mathcal{H}_{k,\ell}$, where $2^k = U$ is known and $\ell = \lceil n^{1/4} \rceil$. Clearly, 2^ℓ can be calculated in time $O(\ell) = O(n^{1/4})$. The values $h(x)$, for all $x \in S$, can be computed in time $O(|S|) = O(n)$; according to Lemma 2.6 h is 1–1 on S with probability at least $1 - n^2/2^{n^{1/4}}$, which is bounded below by $1 - 2^{-n^{1/5}}$ if n is large enough. The deterministic duplicate-grouping algorithm runs in linear time and space, since the size of the integer domain is linear. Therefore the whole algorithm requires linear time and space, and it is exponentially reliable since all the subroutines used are exponentially reliable.

Since the hashing subroutines do not move the elements and both deterministic duplicate-grouping algorithms of Section 2.1 rearrange the elements stably, the whole algorithm is stable. The hashing scheme of Bast and Hagerup is conservative. The justification that the other parts of the algorithm are conservative is straightforward. ∎

Remark 2.12 As concerns reliability, Theorem 2.11 is theoretically stronger than Theorem 2.7, but the program based on the former result will be much more complicated. Moreover, n must be very large before the algorithm of Theorem 2.11 is actually significantly more reliable than that of Theorem 2.7.

In Theorems 2.7 and 2.11 we assumed U to be known. If this is not the case, we have to compute a power of 2 larger than U. Such a number can be obtained by repeated squaring, simply computing 2^{2^i}, for $i = 0, 1, 2, 3, \ldots$, until the first number larger than U is encountered. This takes $O(\log\log U)$ time. Observe also that the largest number manipulated will be at most quadratic in U. Another alternative is to accept both LOG_2 and EXP_2 among the unit-time operations and to use them to compute $2^{\lceil \log_2 U \rceil}$. As soon as the required power of 2 is available, the algorithms described above can be used. Thus, Theorem 2.11 can be extended as follows (the same holds for Theorem 2.7, but only with polynomial reliability).

Theorem 2.13 *The duplicate-grouping problem for a multiset of n integers in the range $\{0, \ldots, U-1\}$ can be solved stably by a conservative randomized algorithm that needs $O(n)$ space and*

(1) *$O(n)$ time on a unit-cost RAM with operations from $\{+, -, *, \text{DIV}, \text{LOG}_2, \text{EXP}_2\}$; or*

(2) *$O(n + \log\log U)$ time on a unit-cost RAM with operations from $\{+, -, *, \text{DIV}\}$.*

The probability that the time bound is exceeded is $2^{-n^{\Omega(1)}}$. ∎

2.5 Randomized duplicate grouping for d-tuples

In the context of the closest-pair problem, the duplicate-grouping problem arises not for multisets of integers from $\{0, \ldots, U-1\}$, but for multisets of d-tuples of integers from $\{0, \ldots, U-1\}$, where d is the dimension of the space under consideration. Even if d is not constant, our algorithms are easily adapted to this situation with a very limited loss of performance. The simplest possibility would be to transform each d-tuple into an integer in the range $\{0, \ldots, U^d - 1\}$ by concatenating the binary representations of the d components, but this would require handling (e. g., multiplying) numbers of around $d \log_2 U$ bits, which may be undesirable. In the proof of the following theorem we describe a different method, which keeps the components of the d-tuples separate and thus deals with numbers of $O(\log U)$ bits only, independently of d.

Theorem 2.14 *Theorems 2.7, 2.11, and 2.13 remain valid if "multiset of n integers" is replaced by "multiset of n d-tuples of integers" and both the time bounds and the probability bounds are multiplied by a factor of d.*

Proof. It is sufficient to indicate how the algorithms described in the proofs of Theorems 2.7 and 2.11 can be extended to accommodate d-tuples. Assume that an array S containing n d-tuples of integers in the range $\{0, \ldots, U-1\}$ is given as input. We proceed in phases $d' = 1, \ldots, d$. In phase d', the entries of S (in the order produced by the previous phase or in the initial order if $d' = 1$) are grouped with respect to component d' by using the method described in the proofs of Theorem 2.7 and 2.11. (In the case of Theorem 2.7, the same hash function should be used for all phases d', in order to avoid using more than $\log_2 U$ random bits.) Even though the d-tuples are rearranged with respect to their hash values, the reordering is always done stably, no matter whether radix sort (Fact 2.1) or the naive deterministic duplicate-grouping algorithm (Lemma 2.3) is employed. This observation allows us to show by induction on d' that after phase d' the d-tuples are grouped stably according to components $1, \ldots, d'$, which establishes the correctness of the algorithm. The time and probability bounds are obvious. ■

3 A randomized closest-pair algorithm

In this section we describe a variant of the random-sampling algorithm of Rabin [27] for solving the closest-pair problem, complete with all details concerning the hashing procedure. For the sake of clarity, we provide a detailed description for the two-dimensional case only.

Let us first define the notion of "grids" in the plane, which is central to the algorithm (and which generalizes easily to higher dimensions). For all $\delta > 0$, a *grid G with resolution δ*, or briefly a *δ-grid G*, consists of two infinite sets of equidistant lines, one parallel to the x-axis, the other parallel to the y-axis, where the distance between two neighboring lines is δ. In precise terms, G is the set

$$\Big\{(x, y) \in \mathbb{R}^2 \;\Big|\; |x - x_0|, |y - y_0| \in \delta \cdot \mathbb{Z}\Big\},$$

for some "origin" $(x_0, y_0) \in \mathbb{R}^2$. The grid G partitions $\mathbb{R}^2$ into disjoint regions called *cells* of G, two points (x, y) and (x', y') being in the same cell if $\lfloor (x - x_0)/\delta \rfloor = \lfloor (x' - x_0)/\delta \rfloor$ and $\lfloor (y - y_0)/\delta \rfloor = \lfloor (y' - y_0)/\delta \rfloor$ (that is, G partitions the plane into half-open squares of side length δ).

Let $S = \{p_1, \ldots, p_n\}$ be a multiset of points in the Euclidean plane. We assume that these points are stored in an array $S[1..n]$. Further, let c be a fixed constant with $0 < c < 1/2$, to be specified later. The algorithm for computing a closest pair in S consists of the following steps.

1. Fix a sample size s with $18n^{1/2+c} \leq s = O(n/\log n)$. Choose a sequence $t_1, \ldots, t_s$ of s elements of $\{1, \ldots, n\}$ randomly. Let $T = \{t_1, \ldots, t_s\}$ and let s' denote the number of distinct elements in T. Store the points p_j with $j \in T$ in an array $R[1..s']$ (R may contain duplicates if S does).

2. Deterministically determine the closest-pair distance δ_0 of the sample stored in R. If R contains duplicates, the result is $\delta_0 = 0$, and the algorithm stops.

3. Compute a closest pair among all the input points. For this, draw a grid G with resolution δ_0 and consider the four different grids G_i with resolution $2\delta_0$, for $i = 1, 2, 3, 4$, that overlap G, i. e., that consist of a subset of the lines in G.

   3a. Group together the points of S falling into the same cell of G_i.

   3b. In each group of at least two points, deterministically find a closest pair; finally output an overall closest pair encountered in this process.

In contrast to Rabin's algorithm [27], we need only one sampling. The sample size s should be $\Omega(n^{1/2+c})$, for some fixed c with $0 < c < 1/2$, to guarantee reliability (cf. Section 4) and $O(n/\log n)$ to ensure that the sample can be handled in linear time. A more formal description of the algorithm is given in Fig. 1.

In [27], Rabin did not describe how to group the points in linear time. As a matter of fact, no linear-time duplicate-grouping algorithms were known at the time. Our construction is based on the algorithms given in Section 2. We assume that the procedure "duplicate-grouping" rearranges the points of S so that all points with the same group index, as determined by the grid cells, are stored consecutively. Let $x_{\min}$ ($y_{\min}$) and $x_{\max}$ ($y_{\max}$) be the smallest and largest x-coordinate (y-coordinate) of a point in S. The group index of a point $p = (x, y)$ is

$$\text{group}_{dx,dy,\delta}(p) = \left(\left\lfloor \frac{x + dx - x_{\min}}{\delta} \right\rfloor, \left\lfloor \frac{y + dy - y_{\min}}{\delta} \right\rfloor \right),$$

a pair of numbers of $O(\log((x_{\max} - x_{\min})/\delta))$ and $O(\log((y_{\max} - y_{\min})/\delta))$ bits. To implement this function, we have to preprocess the points to compute the minimum coordinates $x_{\min}$ and $y_{\min}$.

The correctness of the procedure "randomized-closest-pair" follows from the fact that, since δ_0 is an upper bound on the minimum distance between two points of the multiset S, a closest pair falls into the same cell in at least one of the shifted $2\delta_0$-grids.

Remark 3.1 When computing the distances we have assumed implicitly that the square-root operation is available. However, this is not really necessary. In Step 2 of the algorithm we could calculate the distance δ_0 of a closest pair p_a, p_b of the sample using the Manhattan metric L_1 instead of the Euclidean metric L_2. In Step 3b of the algorithm we could compare the squares of the L_2 distances instead of the actual distances. Since even with this change δ_0 is an upper bound on the L_2-distance of a closest pair, the algorithm will still be correct; on the other hand, the running-time estimate for Step 3, as given in the next section, does not change. (See the analysis of Step 3b following Corollary 4.4.) The tricks just mentioned suffice for showing that the closest-pair algorithm can be made to work for any fixed L_p metric without computing pth roots, if p is a positive integer or ∞.

```
procedure randomized-closest-pair(modifies S: array[1..n] of points)
                                  returns(a pair of points)
% Step 1. Take a random sample of size at most s from the multiset S.
t[1..s] := a random sequence of s indices in [1..n]
% Eliminate repetitions in t[1..s]; store the chosen points in R.
for j := 1 to s do
    T[t[j]] := true
s' := 0
for j := 1 to s do
    if T[t[j]] then
        s' := s' + 1
        R[s'] := S[t[j]]
        T[t[j]] := false
% Step 2. Deterministically compute a closest pair within the random sample.
(p_a, p_b) := deterministic-closest-pair(R[1..s'])
δ_0 := dist(p_a, p_b)                    % dist is the distance function.
if δ_0 > 0 then
    % Step 3. Consider the four overlapping grids.
    for dx, dy ∈ {0, δ_0} do
        % Step 3a. Group the points.
        duplicate-grouping(S[1..n], group_{dx,dy,2δ_0})
        % Step 3b. In each group find a closest pair.
        j := 0
        while j < n do
            i := j + 1
            j := i
            while j < n and group_{dx,dy,2δ_0}(S[i]) = group_{dx,dy,2δ_0}(S[j + 1]) do
                j := j + 1
            if i ≠ j then
                (p_c, p_d) := deterministic-closest-pair(S[i..j])
                if dist(p_c, p_d) < dist(p_a, p_b) then
                    (p_a, p_b) := (p_c, p_d)
return (p_a, p_b)
```

Figure 1: A formal description of the closest-pair algorithm.

Remark 3.2 The randomized closest-pair algorithm generalizes naturally to any d-dimensional space. Note that while two shifts (by 0 and δ_0) of $2\delta_0$-grids are needed in the one-dimensional case, in the two-dimensional case 4 and in the d-dimensional case 2^d shifted grids must be taken into account.

Remark 3.3 For implementing the procedure "deterministic-closest-pair" any of a number of algorithms can be used. Small input sets are best handled by the "brute-force" algorithm, which calculates the distances between all $n(n-1)/2$ pairs of points; in particular, all calls to "deterministic-closest-pair" in Step 3b are executed in this way. For larger input sets, in particular, for the call to "deterministic-closest-pair" in Step 2, we use an asymptotically faster algorithm. For different numbers d of dimensions various algorithms are available. In the *one-dimensional* case the closest-pair problem can be solved by sorting the points and finding the minimum distance between two consecutive points. In the *two-dimensional* case one can use the simple plane-sweep algorithm of Hinrichs et al. [17]. In the *multi-dimensional* case, the divide-and-conquer algorithm of Bentley and Shamos [7] and the incremental algorithm of Schwarz et al. [30] are applicable. Assuming d to be constant, all the algorithms mentioned above run in $O(n \log n)$ time and $O(n)$ space. One should be aware, however, that the complexity depends heavily on d.

4 Analysis of the closest-pair algorithm

In this section, we prove that the algorithm given in Section 3 has linear time complexity with high probability. Again, we treat only the two-dimensional case in detail. Time bounds for most parts of the algorithm were established in previous sections or are immediately clear: Step 1 of the algorithm (taking the sample of size $s' \leq s$) obviously uses $O(s)$ time. Since we assumed that $s = O(n/\log n)$, no more than $O(n)$ time is consumed in Step 2 for finding a closest pair within the sample (see Remark 3.3). The complexity of the grouping performed in Step 3a was analyzed in Section 2. In order to implement the function $group_{dx,dy,\delta}$, which returns the group indices, we need some preprocessing that takes $O(n)$ time.

It remains only to analyze the cost of Step 3b, where closest pairs are found within each group. It will be shown that a sample of size $s \geq 18n^{1/2+c}$, for any fixed c with $0 < c < 1/2$, guarantees $O(n)$-time performance with a failure probability of at most 2^{-n^c}. This holds even if a closest pair within each group is computed by the brute-force algorithm (see Remark 3.3). On the other hand, if the sampling procedure is modified in such a way that only a few 4-wise independent sequences are used to generate the sampling indices $t_1, \ldots, t_s$, linear running time will still be guaranteed with probability $1-O(n^{-\alpha})$, for some constant α, while the number of random bits needed is drastically reduced.

The analysis is complicated by the fact that points may occur repeatedly in the multiset $S = \{p_1, \ldots, p_n\}$. Of course, the algorithm will return two identical points p_a and p_b in this case, and the minimum distance is 0. Note that in

Rabin's paper [27] as well as in that of Khuller and Matias [19], the input points are assumed to be distinct.

Adapting a notion from [27], we first define what it means that there are "many" duplicates and show that in this case the algorithm runs fast. The longer part of the analysis then deals with the situation where there are few or no duplicate points. For reasons of convenience we will assume throughout the analysis that $n \geq 800$.

For a finite (multi)set S and a partition $D = (S_1, \ldots, S_m)$ of S into nonempty subsets, let

$$N(D) = \sum_{\mu=1}^{m} \frac{1}{2}|S_\mu| \cdot (|S_\mu| - 1),$$

which is the number of (unordered) pairs of elements of S that lie in the same set S_μ of the partition. In the case of the natural partition D_S of the multiset $S = \{p_1, \ldots, p_n\}$, where each class consists of all copies of one of the points, we use the following abbreviation:

$$N(S) = N(D_S) = |\{\{i, j\} \mid 1 \leq i < j \leq n \text{ and } p_i = p_j\}|.$$

We first consider the case where $N(S)$ is large; more precisely, we assume for the time being that $N(S) \geq n$. In Section B of the appendix it is proved that under this assumption, if we pick a sample of somewhat more than $\sqrt{n}$ random elements of S, with high probability the sample will contain at least two equal points. More precisely, Corollary B.2 shows that the $s \geq 18n^{1/2+c}$ sample points chosen in Step 1 of the algorithm will contain two equal points with probability at least $1 - 2^{-n^c}$. The deterministic closest-pair algorithm invoked in Step 2 will identify one such pair of duplicates and return $\delta_0 = 0$; at this point the algorithm terminates, having used only linear time.

For the remainder of this section we assume that there are not too many duplicate points, that is, that $N(S) < n$. In this case, we may follow the argument from Rabin's paper. If G is a grid in the plane, then G induces a partition $D_{S,G}$ of the multiset S into disjoint subsets $S_1, \ldots, S_m$ (with duplicates)—two points of S are in the same subset of the partition if and only if they fall into the same cell of G. As in the special case of $N(S)$ above, we are interested in the number

$$N(S, G) = N(D_{S,G}) = |\{\{i, j\} \mid p_i \text{ and } p_j \text{ lie in the same cell of the grid } G\}|.$$

This notion, which was also used in Rabin's analysis [27], expresses the work done in Step 3b when the subproblems are solved by the brute-force algorithm.

Lemma 4.1 [27] *Let S be a multiset of n points in the plane. Further, let G be a grid with resolution δ, and let G' be one of the four grids with resolution 2δ that overlap G. Then $N(S, G') \leq 4N(S, G) + \frac{3}{2}n$.*

Proof. We consider 4 cells of G whose union is one cell of G'. Assume that these 4 cells contain k_1, k_2, k_3, and k_4 points from S (with duplicates), respectively. The

contribution of these cells to $N(S,G)$ is $b = \frac{1}{2}\sum_{i=1}^{4} k_i(k_i-1)$. The contribution of the one (larger) cell to $N(S,G')$ is $\frac{1}{2}k(k-1)$, where $k = \sum_{i=1}^{4} k_i$. We want to give an upper bound on $\frac{1}{2}k(k-1)$ in terms of b.

The function $x \mapsto x(x-1)$ is convex in $[0,\infty)$. Hence

$$\tfrac{1}{4}k\left(\tfrac{1}{4}k-1\right) \le \tfrac{1}{4}\sum_{i=1}^{4} k_i(k_i-1) = \tfrac{1}{2}b.$$

This implies

$$\tfrac{1}{2}k(k-1) = \tfrac{1}{2}k(k-4) + \tfrac{3}{2}k \le 8\cdot\tfrac{1}{4}k\left(\tfrac{1}{4}k-1\right) + \tfrac{3}{2}k \le 4\cdot b + \tfrac{3}{2}k.$$

Summing the last inequality over all cells of G' yields the desired inequality $N(S,G') \le 4N(S,G) + \frac{3}{2}n$. ∎

Remark 4.2 In the case of d-dimensional space, this calculation can be carried out in exactly the same way; this results in the estimate $N(S,G') \le 2^d N(S,G) + \frac{1}{2}(2^d-1)n$.

Corollary 4.3 *Let S be a multiset of n points that satisfies $N(S) < n$. Then there is a grid G^* with $n \le N(S,G^*) < 5.5n$.*

Proof. We start with a grid G so fine that no cell of the grid contains two distinct points in S. Then, obviously, $N(S,G) = N(S) < n$. By repeatedly doubling the grid size as in Lemma 4.1 until $N(S,G') \ge n$ for the first time, we find a grid G^* satisfying the claim. ∎

Corollary 4.4 *Let S be a multiset of size n and let G be a grid with resolution δ. Further, let G' be an arbitrary grid with resolution at most δ. Then $N(S,G') \le 16N(S,G) + 6n$.*

Proof. Let G_i, for $i = 1,2,3,4$, be the four different grids with resolution 2δ that overlap G. Each cell of G' is completely contained in some cell of at least one of the grids G_i. Thus, the sets of the partition induced by G' can be divided into four disjoint classes depending on which of the grids G_i covers the corresponding cell completely. Therefore, we have $N(S,G') \le \sum_{i=1}^{4} N(S,G_i)$. Applying Lemma 4.1 and summing up yields $N(S,G') \le 16N(S,G) + 6n$, as desired. ∎

Now we are ready for analyzing Step 3b of the algorithm. As stated above, we assume that $N(S) < n$; hence the existence of some grid G^* as in Corollary 4.3 is ensured. Let $\delta^* > 0$ denote the resolution of G^*.

We apply Corollary B.2 from the appendix to the partition of S (with duplicates) induced by G^* to conclude that with probability at least $1 - 2^{-n^c}$ the random sample taken in Step 1 of the algorithm contains two points from the same cell of G^*. It remains to show that if this is the case then Step 3b of the algorithm takes $O(n)$ time.

Since the real number δ_0 calculated by the algorithm in Step 2 is bounded by the distance of two points in the same cell of G^*, we must have $\delta_0 \le 2\delta^*$. (This

is the case even if in Step 2 the Manhattan metric L_1 is used.) Thus the four grids G_1, G_2, G_3, G_4 used in Step 3 have resolution $2\delta_0 \leq 4\delta^*$. We form a new conceptual grid G^{**} with resolution $4\delta^*$ by omitting all but every fourth line from G^*. By the inequality $N(S, G^*) < 5.5n$ (Corollary 4.3) and a double application of Lemma 4.1, we obtain $N(S, G^{**}) = O(n)$. The resolution $4\delta^*$ of the grid G^{**} is at least $2\delta_0$. Hence we may apply Corollary 4.4 to obtain that the four grids G_1, G_2, G_3, G_4 used in Step 3 of the algorithm satisfy $N(S, G_i) = O(n)$, for $i = 1, 2, 3, 4$. But obviously the running time of Step 3b is $O(\sum_{i=1}^{4}(N(S, G_i)+n))$; by the above, this bound is linear in n. This finishes the analysis of the cost of Step 3b.

It is easy to see that Corollaries 4.3 and 4.4 as well as the analysis of Step 3b generalize from the plane to any fixed dimension d. Combining the discussion above with Theorem 2.13, we obtain the following.

Theorem 4.5 *The closest-pair problem for a multiset of n points in d-dimensional space, where $d \geq 1$ is a fixed integer, can be solved by a randomized algorithm that needs $O(n)$ space and*

(1) *$O(n)$ time on a real RAM with operations from $\{+, -, *, \text{DIV}, \text{LOG}_2, \text{EXP}_2\}$; or*

(2) *$O(n+\log\log(\delta_{\max}/\delta_{\min}))$ time on a real RAM with operations from $\{+, -, *, \text{DIV}\}$,*

where $\delta_{\max}$ and $\delta_{\min}$ denote the maximum and the minimum distance between any two distinct points, respectively. The probability that the time bound is exceeded is $2^{-n^{\Omega(1)}}$.

Proof. The running time of the randomized closest-pair algorithm is dominated by that of Step 3a. The group indices used in Step 3a are d-tuples of integers in the range $\{0, \ldots, \lceil\delta_{\max}/\delta_{\min}\rceil\}$. By Theorem 2.14, parts (1) and (2) of the theorem follow directly from the corresponding parts of Theorem 2.13. Since all the subroutines used finish within their respective time bounds with probability $1-2^{-n^{\Omega(1)}}$, the same is true for the whole algorithm. The amount of space required is obviously linear. ∎

In the situation of Theorem 4.5, if the coordinates of the input points happen to be integers drawn from a range $\{0, \ldots, U-1\}$, we can replace the real RAM by a conservative unit-cost RAM with integer operations; the time bound of part (2) then becomes $O(n+\log\log U)$. The number of random bits used by either version of the algorithm is quite large, namely essentially as large as possible with the given running time. Even if the number of random bits used is severely restricted, we can still retain an algorithm that is polynomially reliable.

Theorem 4.6 *Let $\alpha, d \geq 1$ be arbitrary fixed integers. The closest-pair problem for a multiset of n points in d-dimensional space can be solved by a randomized algorithm with the time and space requirements stated in Theorem 4.5 that uses only $O(\log n + \log(\delta_{\max}/\delta_{\min}))$ random bits (or $O(\log n + \log U)$ random bits for integer input coordinates in the range $\{0, \ldots, U-1\}$), and that exceeds the time bound with probability $O(n^{-\alpha})$.*

Proof. We let $s = 16\alpha \cdot \lceil n^{3/4} \rceil$ and generate the sequence $t_1, \ldots, t_s$ in the algorithm as the concatenation of 4α independently chosen sequences of 4-independent random values that are approximately uniformly distributed in $\{1, \ldots, n\}$. This random experiment and its properties are described in detail in Corollary B.4 and Lemma B.5 in Section B of the appendix. The time needed is $o(n)$, and the number of random bits needed is $O(\log n)$. The duplicate grouping is performed with the simple method described in Section 2.3. This requires only $O(\log(\delta_{\max}/\delta_{\min}))$ or $O(\log U)$ random bits. The analysis is exactly the same as in the proof of Theorem 4.5, except that Corollary B.4 is used instead of Corollary B.2. ∎

5 Conclusions

We have provided an asymptotically efficient algorithm for computing a closest pair of n points in d-dimensional space. The main idea of the algorithm is to use random sampling in order to reduce the original problem to a collection of duplicate-grouping problems. The performance of the algorithm depends on the operations assumed to be primitive in the underlying machine model. We proved that, with high probability, the running time is $O(n)$ on a real RAM capable of executing the arithmetic operations from $\{+, -, *, \text{DIV}, \text{LOG}_2, \text{EXP}_2\}$ in constant time. Without the operations LOG_2 and EXP_2, the running time increases by an additive term of $O(\log\log(\delta_{\max}/\delta_{\min}))$, where $\delta_{\max}$ and $\delta_{\min}$ denote the maximum and the minimum distance between two distinct points, respectively. When the coordinates of the points are integers in the range $\{0, \ldots, U-1\}$, the running times are $O(n)$ and $O(n + \log\log U)$, respectively. For integer data the algorithm is conservative, i.e., all the numbers manipulated contain $O(\log n + \log U)$ bits.

We proved that the bounds on the running times hold also when the collection of input points contains duplicates. As an immediate corollary of this result we get that the following decision problems, which are often used in lower-bound arguments for geometric problems (see [26]), can be solved as efficiently as the one-dimensional closest-pair problem on the real RAM (Theorems 4.5 and 4.6):

(1) *Element-distinctness problem:* Given n real numbers, decide if any two of them are equal.

(2) *ε-closeness problem:* Given n real numbers and a threshold value $\varepsilon > 0$, decide if any two of the numbers are at distance less than ε from each other.

Finally, we would like to mention practical experiments with our simple duplicate-grouping algorithm. The experiments were conducted by Tomi Pasanen (University of Turku, Finland). He found that the duplicate-grouping algorithm described in Theorem 2.7, which is based on radix sort (with $\alpha = 3$), behaves essentially as well as heapsort. For small inputs ($n < 50\,000$) heapsort was slightly faster, whereas for large inputs heapsort was slightly slower. Randomized quicksort turned out to be much faster than any of these algorithms for all $n \leq 1\,000\,000$. One drawback of the radix-sort algorithm is that it requires extra memory space

for linking the duplicates, whereas heapsort (as well as in-place quicksort) does not require any extra space. One should also note that in some applications the word length of the actual machine can be restricted to, say, 32 bits. This means that when $n > 2^{11}$ and $\alpha = 3$, the hash function $h \in \mathcal{H}_{k,\ell}$ (see the proof of Theorem 2.7) is not needed for collapsing the universe; radix sort can be applied directly. Therefore the integers must be long before the full power of our methods comes into play.

Acknowledgements

We would like to thank Ivan Damgård for his comments concerning Lemma A.1 and Tomi Pasanen for his assistance in evaluating the practical efficiency of the duplicate-grouping algorithm. The question of whether the class of multiplicative hash functions is universal was posed to the first author by Ferri Abolhassan and Jörg Keller. We also thank Kurt Mehlhorn for useful comments on this universal class and on the issue of 4-independent sampling.

References

[1] A. Aggarwal, H. Edelsbrunner, P. Raghavan, and P. Tiwari, Optimal time bounds for some proximity problems in the plane, *Inform. Process. Lett.* **42** (1992), 55–60.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, "The Design and Analysis of Computer Algorithms", Addison-Wesley, Reading, 1974.

[3] A. Andersson, T. Hagerup, S. Nilsson, and R. Raman, Sorting in linear time?, *in* "Proc. 27th Annual ACM Symposium on the Theory of Computing", pp. 427–436, Association for Computing Machinery, New York, 1995.

[4] H. Bast and T. Hagerup, Fast and reliable parallel hashing, *in* "Proc. 3rd Annual ACM Symposium on Parallel Algorithms and Architectures", pp. 50–61, Association for Computing Machinery, New York, 1991.

[5] P. Beauchemin, G. Brassard, C. Crépeau, C. Goutier, and C. Pomerance, The generation of random numbers that are probably prime, *J. Cryptology* **1** (1988), 53–64.

[6] M. Ben-Or, Lower bounds for algebraic computation trees, *in* "Proc. 15th Annual ACM Symposium on Theory of Computing", pp. 80–86, Association for Computing Machinery, New York, 1983.

[7] J. L. Bentley and M. I. Shamos, Divide-and-conquer in multidimensional space, *in* "Proc. 8th Annual ACM Symposium on Theory of Computing", pp. 220–230, Association for Computing Machinery, New York, 1976.

[8] J. L. Carter and M. N. Wegman, Universal classes of hash functions, *J. Comput. System Sci.* **18** (1979), 143–154.

[9] B. Chor and O. Goldreich, On the power of two-point based sampling, *J. Complexity* **5** (1989), 96–106.

[10] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, "Introduction to Algorithms", The MIT Press, Cambridge, 1990.

[11] I. Damgård, P. Landrock, and C. Pomerance, Average case error estimates for the strong probable prime test, *Math. Comp.* **61** (1993), 177–194.

[12] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, and R. E. Tarjan, Dynamic perfect hashing: Upper and lower bounds, *SIAM J. Comput.* **23** (1994), 738–761.

[13] M. Dietzfelbinger and F. Meyer auf der Heide, Dynamic hashing in real time, *in* "Informatik · Festschrift zum 60. Geburtstag von Günter Hotz" (J. Buchmann, H. Ganzinger, and W. J. Paul, Eds.), Teubner-Texte zur Informatik, Band 1, pp. 95–119, B. G. Teubner, Stuttgart, 1992.

[14] M. L. Fredman, J. Komlós and E. Szemerédi, Storing a sparse table with $O(1)$ worst case access time, *J. Assoc. Comput. Mach.* **31** (1984), 538–544.

[15] S. Fortune and J. Hopcroft, A note on Rabin's nearest-neighbor algorithm, *Inform. Process. Lett.* **8** (1979), 20–23.

[16] M. Golin, R. Raman, C. Schwarz, and M. Smid, Simple randomized algorithms for closest pair problems, *Nordic J. Comput.* **2** (1995), 3–27.

[17] K. Hinrichs, J. Nievergelt, and P. Schorn, Plane-sweep solves the closest pair problem elegantly, *Inform. Process. Lett.* **26** (1988), 255–261.

[18] J. Katajainen and M. Lykke, "Experiments with universal hashing", Technical Report 96/8, Dept. of Computer Science, Univ. of Copenhagen, Copenhagen, 1996.

[19] S. Khuller and Y. Matias, A simple randomized sieve algorithm for the closest-pair problem, *Inform. and Comput.* **118** (1995), 34–37.

[20] D. Kirkpatrick and S. Reisch, Upper bounds for sorting integers on random access machines, *Theoret. Comput. Sci.* **28** (1984), 263–276.

[21] D. E. Knuth, "The Art of Computer Programming, Vol. 3: Sorting and Searching", Addison-Wesley, Reading, 1973.

[22] Y. Mansour, N. Nisan, and P. Tiwari, The computational complexity of universal hashing, *in* "Proc. 22nd Annual ACM Symposium on Theory of Computing", pp. 235–243, Association for Computing Machinery, New York, 1990.

[23] Y. Matias and U. Vishkin, "On parallel hashing and integer sorting", Technical Report UMIACS–TR–90–13.1, Inst. for Advanced Computer Studies, Univ. of Maryland, College Park, 1990. (Journal version: *J. Algorithms* **12** (1991), 573–606.)

[24] K. Mehlhorn, "Data Structures and Algorithms, Vol. 1: Sorting and Searching", Springer-Verlag, Berlin, 1984.

[25] G. L. Miller, Riemann's hypothesis and tests for primality, *J. Comput. System Sci.* **13** (1976), 300–317.

[26] F. P. Preparata and M. I. Shamos, "Computational Geometry: An Introduction", Springer-Verlag, New York, 1985.

[27] M. O. Rabin, Probabilistic algorithms, *in* "Algorithms and Complexity: New Directions and Recent Results" (J. F. Traub, Ed.), pp. 21–39, Academic Press, New York, 1976.

[28] M. O. Rabin, Probabilistic algorithm for testing primality, *J. Number Theory* **12** (1980), 128–138.

[29] R. Raman, Priority queues: small, monotone and trans-dichotomous, *in* "Proc. 4th Annual European Symposium on Algorithms", Lecture Notes in Comput. Sci. **1136**, pp. 121–137, Springer, Berlin, 1996.

[30] C. Schwarz, M. Smid, and J. Snoeyink, An optimal algorithm for the on-line closest-pair problem, *in* "Proc. 8th Annual Symposium on Computational Geometry", pp. 330–336, Association for Computing Machinery, New York, 1992.

[31] W. Sierpiński, "Elementary Theory of Numbers", Second English Edition (A. Schinzel, Ed.), North-Holland, Amsterdam, 1988.

[32] A. C.-C. Yao, Lower bounds for algebraic computation trees with integer inputs, *SIAM J. Comput.* **20** (1991), 655–668.

A Generating primes

In this section we provide a proof of Lemma 2.9. The main idea is expressed in the proof of the following lemma.

Lemma A.1 *There is a randomized algorithm that, for any given integer $m \geq 2$, returns an integer p with $m < p \leq 2m$ such that the following holds: the running time is $O((\log m)^4)$, and the probability that p is not prime is at most $1/m$.*

Proof. The heart of the construction is the randomized primality test due to Miller [25] and Rabin [28] (for a description and an analysis see, e. g., [10, pp. 839 ff.]). If an arbitrary number x of b bits is given to the test as an input, then the following holds:

(a) If x is prime, then **Prob**(the result of the test is "prime") $= 1$;

(b) if x is composite, then **Prob**(the result of the test is "prime") $\leq 1/4$;

(c) performing the test once requires $O(b)$ time, and all numbers manipulated in the test are $O(b)$ bits long.

By repeating the test t times, the reliability of the result can be increased such that for composite x we have

$$\textbf{Prob}(\text{the result of the test is "prime"}) \leq 1/4^t.$$

In order to generate a "probable prime" that is greater than m we use a random sampling algorithm. We select s (to be specified later) integers from the interval $\{m+1, \ldots, 2m\}$ at random. Then these numbers are tested one by one until the result of the test is "prime". If no such result is obtained the number $m+1$ is returned.

The algorithm fails to return a prime number (1) if there is no prime among the numbers in the sample, or (2) if one of the composite numbers in the sample is accepted by the primality test. We estimate the probabilities of these events.

It is known that the function $\pi(x) = |\{p|\, p \leq x \text{ and } p \text{ is prime}\}|$, defined for any real number x, satisfies

$$\pi(2n) - \pi(n) > \frac{n}{3\ln(2n)},$$

for all integers $n > 1$. (For a complete proof of this fact, also known as the inequality of Finsler, see [31, Sections 3.10 and 3.14].) That is, the number of primes in the set $\{m+1, \ldots, 2m\}$ is at least $m/(3\ln(2m))$. We choose

$$s = s(m) = \lceil 3(\ln(2m))^2 \rceil$$

and

$$t = t(m) = \max\{\lceil \log_2 s(m) \rceil, \lceil \log_2(2m) \rceil\}.$$

(Note that $t(m) = O(\log m)$.) Then the probability that the random sample contains no prime at all is bounded by

$$\left(1 - \frac{1}{3\ln(2m)}\right)^s \leq \left(\left(1 - \frac{1}{3\ln(2m)}\right)^{3\ln(2m)}\right)^{\ln(2m)} < e^{-\ln(2m)} = \frac{1}{2m}.$$

The probability that one of the at most s composite numbers in the sample will be accepted is smaller than

$$s(m) \cdot (1/4)^t \le s(m) \cdot 2^{-\log_2 s(m)} \cdot 2^{-\log_2(2m)} = \frac{1}{2m}.$$

Summing up, the failure probability of the algorithm is at most $2 \cdot (1/(2m)) = 1/m$, as claimed. If m is a b-bit number, the time required is $O(s \cdot t \cdot b)$, that is, $O((\log m)^4)$. ∎

Remark A.2 The problem of generating primes is discussed in greater detail by Damgård et al. [11]. Their analysis shows that the proof of Lemma A.1 is overly pessimistic. Therefore, without sacrificing the reliability, the sample size s and/or the repetition count t can be decreased; in this way considerable savings in the running time are possible.

Lemma 2.9 *There is a randomized algorithm that, for any given positive integers m and n with $2 \le m \le 2^{\lceil n^{1/4} \rceil}$, returns a number p with $m < p \le 2m$ such that the following holds: the running time is $O(n)$, and the probability that p is not prime is at most $2^{-n^{1/4}}$.*

Proof. We increase the sample size s and the repetition count t in the algorithm of Lemma A.1 above, as follows:

$$s = s(m, n) = 6 \cdot \lceil \ln(2m) \rceil \cdot \lceil n^{1/4} \rceil$$

and

$$t = t(m, n) = 1 + \max\{\lceil \log_2 s(m, n) \rceil, \lceil n^{1/4} \rceil\}.$$

As above, the failure probability is bounded by the sum of the following two terms:

$$\left(1 - \frac{1}{3\ln(2m)}\right)^{s(m,n)} < e^{-2\lceil n^{1/4} \rceil} < 2^{-1-n^{1/4}}$$

and

$$s(m, n) \cdot (1/4)^{t(m,n)} \le 2^{-(1+\lceil n^{1/4} \rceil)} \le 2^{-1-n^{1/4}}.$$

This proves the bound $2^{-n^{1/4}}$ on the failure probability. The running time is

$$O(s \cdot t \cdot \log m) = O((\log m) \cdot n^{1/4} \cdot (\log\log m + \log n + n^{1/4}) \cdot \log m) = O(n).$$

∎

B Random sampling in partitions

In this section we deal with some technical details of the analysis of the closest-pair algorithm. For a finite set S and a partition $D = (S_1, \ldots, S_m)$ of S into nonempty subsets, let

$$P(D) = \{\pi \subseteq S \mid |\pi| = 2 \wedge \exists \mu \in \{1, \ldots, m\} : \pi \subseteq S_\mu\}.$$

Note that the quantity $N(D)$ defined in Section 4 equals $|P(D)|$. For the analysis of the closest-pair algorithm, we need the following technical fact: If $N(D)$ is linear in n and more than $8\sqrt{n}$ elements are chosen at random from S, then with a probability that is not too small two elements from the same subset of the partition are picked. A similar lemma was proved by Rabin [27, Lemma 6]. In Section B.1 we give a totally different proof, resting on basic facts from probability theory (viz., Chebyshev's inequality), which may make it more conspicuous why the lemma is true than Rabin's proof. Further, it will turn out that full independence of the elements in the random sample is not needed, but rather that 4-wise independence is sufficient. This observation is crucial for a version of the closest-pair algorithm that uses only few random bits. The technical details are given in Section B.2.

B.1 The sampling lemma

Lemma B.1 *Let n, m and s be positive integers, let S be a set of size $n \geq 800$, let $D = (S_1, \ldots, S_m)$ be a partition of S into nonempty subsets with $N(D) \geq n$, and assume that s random elements $t_1, \ldots, t_s$ are drawn independently from the uniform distribution over S. Then if $s \geq 8\sqrt{n}$,*

$$\mathbf{Prob}\Big(\exists i, j \in \{1, \ldots, s\} \exists \mu \in \{1, \ldots, m\} : t_i \neq t_j \wedge t_i, t_j \in S_\mu\Big) > 1 - \frac{4\sqrt{n}}{s}. \quad \text{(B.1)}$$

Proof. We first note that we may assume, without loss of generality, that

$$n \leq N(D) \leq 1.1n. \quad \text{(B.2)}$$

To see this, assume that $N(D) > 1.1n$ and consider a process of repeatedly refining D by splitting off an element x in a largest set in D, i.e., by making x into a singleton set. As long as D contains a set of size $\sqrt{2n} + 2$ or more, the resulting partition D' still has $N(D') \geq n$. On the other hand, splitting off an element from a set of size less that $\sqrt{2n}+2$ changes N by less than $\sqrt{2n}+1 = \sqrt{200/n} \cdot 0.1n + 1$, which for $n \geq 800$ is at most $0.1n$. Hence if we stop the process with the first partition D' with $N(D') \leq 1.1n$, we will still have $N(D') \geq n$. Since D' is a refinement of D, we have for all i and j that

$$\begin{aligned} &t_i \text{ and } t_j \text{ are contained in the same set } S'_\mu \text{ of } D' \\ &\quad \Rightarrow t_i \text{ and } t_j \text{ are contained in the same set } S_\mu \text{ of } D; \end{aligned}$$

thus, it suffices to prove (B.1) for D'.

We define random variables $X^{\pi}_{i,j}$, for $\pi \in P(D)$ and $1 \leq i < j \leq s$, as follows:

$$X^{\pi}_{i,j} := \begin{cases} 1 & \text{if } \{t_i, t_j\} = \pi, \\ 0 & \text{otherwise.} \end{cases}$$

Further, we let

$$X = \sum_{\pi \in P(D)} \sum_{1 \leq i < j \leq s} X^{\pi}_{i,j}.$$

Clearly, by the definition of $P(D)$,

$$X = |\{(i,j) \mid 1 \leq i < j \leq s \wedge t_i \neq t_j \wedge t_i, t_j \in S_\mu \text{ for some } \mu\}| \geq 0.$$

Thus, to establish (B.1), we only have to show that

$$\mathbf{Prob}(X = 0) < \frac{4\sqrt{n}}{s}.$$

For this, we estimate the expectation $\mathbf{E}(X)$ and the variance $\mathbf{Var}(X)$ of the random variable X, with the intention of applying Chebyshev's inequality:

$$\mathbf{Prob}\Big(|X - \mathbf{E}(X)| \geq t\Big) \leq \frac{\mathbf{Var}(X)}{t^2}, \quad \text{for all } t > 0. \tag{B.3}$$

(For another, though simpler, application of Chebyshev's inequality in a similar context see [9]).

First note that for each $\pi = \{x, y\} \in P(D)$ and $1 \leq i < j \leq s$ the following holds:

$$\mathbf{E}(X^{\pi}_{i,j}) = \mathbf{Prob}(t_i = x \wedge t_j = y) + \mathbf{Prob}(t_i = y \wedge t_j = x) = \frac{2}{n^2}. \tag{B.4}$$

Thus,

$$\begin{aligned} \mathbf{E}(X) &= \sum_{\pi \in P(D)} \sum_{1 \leq i < j \leq s} \mathbf{E}(X^{\pi}_{i,j}) \qquad (B.5) \\ &= |P(D)| \cdot \binom{s}{2} \cdot \frac{2}{n^2} = N(D) \cdot \frac{s^2}{n^2} \cdot \left(1 - \frac{1}{s}\right). \end{aligned}$$

By assumption, $s \geq 8\sqrt{n} \geq 8\sqrt{800}$, so that $1 - 1/s \geq 1/1.01$. Let $\alpha = s/\sqrt{n}$. Using the assumption $N(D) \geq n$, we get from (B.5) that

$$\mathbf{E}(X) \geq \frac{\alpha^2}{1.01}. \tag{B.6}$$

Next we derive an upper bound on the variance of X. With the (standard) notation

$$\mathbf{Cov}(X^{\pi}_{i,j}, X^{\pi'}_{i',j'}) = \mathbf{E}(X^{\pi}_{i,j} \cdot X^{\pi'}_{i',j'}) - \mathbf{E}(X^{\pi}_{i,j}) \cdot \mathbf{E}(X^{\pi'}_{i',j'})$$

we may write

$$\mathbf{Var}(X) = \mathbf{E}(X^2) - (\mathbf{E}(X))^2 = \sum_{\pi,\pi' \in P(D)} \sum_{\substack{1 \le i < j \le s \\ 1 \le i' < j' \le s}} \mathbf{Cov}(X^{\pi}_{i,j}, X^{\pi'}_{i',j'}). \qquad (B.7)$$

We split the summands $\mathbf{Cov}(X^{\pi}_{i,j}, X^{\pi'}_{i',j'})$ occurring in this sum into several classes and estimate the contribution to $\mathbf{Var}(X)$ of the summands in each of these classes. For all except the first class, we use the simple bound

$$\mathbf{Cov}(X^{\pi}_{i,j}, X^{\pi'}_{i',j'}) \le \mathbf{E}(X^{\pi}_{i,j} \cdot X^{\pi'}_{i',j'}) = \mathbf{Prob}(X^{\pi}_{i,j} = X^{\pi'}_{i',j'} = 1).$$

For $i \in \{1, \ldots, s\}$, if $t_i = x \in S$, we will say that i is *mapped to* x. Below we bound the probability that $\{i, j\}$ is mapped onto π, while at the same time $\{i', j'\}$ is mapped onto π'. Let $J = \{i, j, i', j'\}$.

Class 1. $|J| = 4$. In this case the random variables $X^{\pi}_{i,j}$ and $X^{\pi'}_{i',j'}$ are independent, so that $\mathbf{Cov}(X^{\pi}_{i,j}, X^{\pi'}_{i',j'}) = 0$.

Class 2. $|J| = 2$ and $\pi = \pi'$. Now $\mathbf{E}(X^{\pi}_{i,j} \cdot X^{\pi'}_{i',j'}) = \mathbf{E}(X^{\pi}_{i,j})$, so the total contribution to $\mathbf{Var}(X)$ of summands of Class 2 is at most

$$\sum_{\pi \in P(D)} \sum_{1 \le i < j \le s} \mathbf{E}(X^{\pi}_{i,j}) = \mathbf{E}(X).$$

Class 3. $|J| < |\pi \cup \pi'|$. In this case J cannot be mapped onto $\pi \cup \pi'$, so $X^{\pi}_{i,j} \cdot X^{\pi'}_{i',j'} \equiv 0$ and $\mathbf{Cov}(X^{\pi}_{i,j}, X^{\pi'}_{i',j'}) \le 0$.

Since $|J| \in \{2, 3, 4\}$ and $|\pi \cup \pi'| \in \{2, 3, 4\}$, the only case not covered above is $|J| = 3$ and $|\pi \cup \pi'| \in \{2, 3\}$. In order to simplify the discussion of this final case, let us call the single element of $\{i, j\} \cap \{i', j'\}$ the *central domain element*. Correspondingly, if $|\pi \cup \pi'| = 3$, we call the single element of $\pi \cap \pi'$ the *central range element*. The argument proceeds by counting the number of summands of certain kinds as well as estimating the size of each summand.

Class 4. $|J| = 3$ and $|\pi \cup \pi'| \in \{2, 3\}$. The central domain element and the other elements of $\{i, j\}$ and $\{i', j'\}$ can obviously be chosen in no more than s^3 ways.

Class 4a. $|J| = 3$ and $\pi = \pi'$. By definition, $\pi = \pi'$ can be chosen in $N(D)$ ways. Furthermore, $X^{\pi}_{i,j} = X^{\pi}_{i',j'} = 1$ only if the central domain element is mapped to one element of π, while the two remaining elements of J are both mapped to the other element of π, the probability of which is $(2/n)(1/n)(1/n) = 2/n^3$. Altogether, the contribution to $\mathbf{Var}(X)$ of summands of Class 4a is at most $s^3 \cdot N(D) \cdot 2/n^3 \le 2.2s^3/n^2$.

Class 4b. $|J| = 3$ and $|\pi \cup \pi'| = 3$. The set $\pi \cup \pi'$ can be chosen in $\sum_{\mu=1}^{m} \binom{|S_\mu|}{3}$ ways, after which there are three choices for the central range element and two ways of completing π (and, implicitly, π') with one of the remaining elements of $\pi \cup \pi'$. $X^{\pi}_{i,j} = X^{\pi'}_{i',j'} = 1$ only if the central domain element is mapped to the central range element, while the remaining element of $\{i, j\}$ is mapped to the remaining element of π and the remaining element of $\{i', j'\}$ is mapped to the

remaining element of π', the probability of which is $1/n^3$. It follows that the total contribution to $\mathbf{Var}(X)$ of summands of Class 4b is bounded by

$$\left(\sum_{\mu=1}^{m} |S_\mu|(|S_\mu|-1)(|S_\mu|-2)\right)\cdot\left(\frac{s}{n}\right)^3 \le \left(\sum_{\mu=1}^{m}(|S_\mu|-1)^3\right)\cdot\left(\frac{s}{n}\right)^3. \qquad \text{(B.8)}$$

We use the inequality $\sum_{\mu=1}^m a_\mu^3 \le \left(\sum_{\mu=1}^m a_\mu^2\right)^{3/2}$ (a special case of Jensen's inequality, valid for all $a_1,\ldots,a_m \ge 0$) and the assumption (B.2) to bound the right hand side in (B.8) by

$$\left(\sum_{\mu=1}^{m} |S_\mu|(|S_\mu|-1)\right)^{3/2}\cdot\left(\frac{s}{n}\right)^3 \le (2\cdot 1.1n)^{3/2}\cdot\left(\frac{s}{n}\right)^3 = 2.2^{3/2}\cdot\left(\frac{s}{\sqrt{n}}\right)^3 < 3.3\alpha^3.$$

Bounding the contributions of the summands of the various classes to the sum in equation (B.7), we get (using that $n^{1/2}\ge 25$)

$$\begin{aligned}\mathbf{Var}(X) &\le \mathbf{E}(X) + 2.2s^3/n^2 + 3.3\alpha^3 = \mathbf{E}(X) + (2.2n^{-1/2}+3.3)\alpha^3\\ &< \mathbf{E}(X) + 3.5\alpha^3. \end{aligned}\qquad \text{(B.9)}$$

By (B.3) we have

$$\mathbf{Prob}(X=0) \le \mathbf{Prob}\Big(|X-E(X)| \ge E(X)\Big) \le \frac{\mathbf{Var}(X)}{(\mathbf{E}(X))^2};$$

by (B.9) and (B.6) this yields

$$\mathbf{Prob}(X=0) \le \frac{1}{\mathbf{E}(X)} + \frac{3.5\alpha^3}{(\mathbf{E}(X))^2} \le \frac{1.01}{\alpha^2} + \frac{3.5\cdot 1.01^2}{\alpha}.$$

Since $1.01/\alpha + 3.5\cdot 1.01^2 < 4$, we get

$$\mathbf{Prob}(X=0) < \frac{4}{\alpha} = \frac{4\sqrt{n}}{s},$$

as claimed. ∎

In case the size of the chosen subset is much larger than $\sqrt{n}$, the estimate in the lemma can be considerably sharpened.

Corollary B.2 *Let n, m and s be positive integers, let S be a set of size $n \ge 800$, let $D = (S_1,\ldots,S_m)$ be a partition of S into nonempty subsets with $N(D) \ge n$, and assume that s random elements $t_1,\ldots,t_s$ are drawn independently from the uniform distribution over S. Then if $s \ge 9\sqrt{n}$,*

$$\mathbf{Prob}\Big(\exists i,j\in\{1,\ldots,s\}\exists\mu\in\{1,\ldots,m\} : t_i \ne t_j \wedge t_i,t_j \in S_\mu\Big) > 1 - 2^{-s/(18\sqrt{n})}.$$

Proof. Split the sequence $t_1, \ldots, t_s$ into disjoint subsequences of length $s' = \lceil 8\sqrt{n} \rceil \leq 9\sqrt{n}$ each, with fewer than s' elements left over. By Lemma B.1, in each of the corresponding subexperiments the probability that two elements in the same subset S_μ are hit is at least $1 - 4\sqrt{n}/s' \geq \frac{1}{2}$. Since the subexperiments are independent and their number is at least $\lfloor s/(9\sqrt{n}) \rfloor \geq s/(18\sqrt{n})$, the stated event will occur in at least one of them with probability at least $1 - 2^{-s/(18\sqrt{n})}$. Clearly, this is also a lower bound on the probability that the whole sequence $t_1, \ldots, t_s$ hits two elements from the same S_μ. ∎

B.2 Sampling with few random bits

In this section we show that the effect described in Lemma B.1 can be achieved also with a random experiment that uses very few random bits.

Corollary B.3 *Let n, m, s, S, and D be as in Lemma B.1. Then the conclusion of Lemma B.1 also holds if the s elements $t_1, \ldots, t_s$ are chosen according to a distribution over S that only satisfies the following two conditions:*

(a) the sequence is 4-independent, i. e., for all sets $\{i, j, k, \ell\} \subseteq \{1, \ldots, s\}$ of size 4 the values t_i, t_j, t_k, t_ℓ are independent; and

(b) for all $i \in \{1, \ldots, s\}$ and all $x \in S$ we have

$$\frac{1-\varepsilon}{n} < \mathbf{Prob}(t_i = x) < \frac{1+\varepsilon}{n},$$

where $\varepsilon = 0.0025$.

Proof. This is proved almost exactly as Lemma B.1. We indicate the slight changes that have to be made. Equation (B.4) is replaced by

$$\mathbf{E}(X_{i,j}^\pi) \geq 2 \cdot \left(\frac{1-\varepsilon}{n}\right)^2 \geq \frac{2(1-2\varepsilon)}{n^2}.$$

Equation (B.5) changes into

$$\mathbf{E}(X) \geq N(D) \cdot \frac{s^2}{n^2} \cdot (1-2\varepsilon) \cdot \left(1 - \frac{1}{s}\right).$$

As $s \geq 8\sqrt{800}$ and $\varepsilon = 0.0025$, we get $(1-2\varepsilon)(1-1/s) \geq 1/1.01$, such that (B.6) remains valid. The contributions to $\mathbf{Var}(X)$ of the summands of the various classes defined in the proof of Lemma B.1 are bounded as follows.

Class 1: The contribution is 0. For justifying this, 4-wise independence is sufficient.

Class 2: $\mathbf{E}(X)$.

Class 3: ≤ 0.

Class 4a: $s^3 \cdot N(D) \cdot (2/n^3) \cdot (1+\varepsilon)^3 \le 2.3s^3/n^2$.

Class 4b: $(2.2n)^{3/2} \cdot (s/n^3) \cdot (1+\varepsilon)^3 \le 3.3\alpha^3$.

Finally, estimate (B.9) is replaced by

$$\mathbf{Var}(x) \le \mathbf{E}(X) + (2.3n^{-1/2} + 3.3)\alpha^3 < \mathbf{E}(X) + 3.5\alpha^3,$$

where we used that $n^{1/2} \ge 25$. The rest of the argument is verbally the same as in the proof of Lemma B.1. ∎

In the random sampling experiment, we can even achieve polynomial reliability with a moderate number of random bits.

Corollary B.4 *In the situation of Lemma B.1, let* $s \ge 4\lceil n^{3/4} \rceil$, *and let* $\alpha \ge 1$ *be an arbitrary integer. If the experiment described in Corollary B.3 is repeated independently* 4α *times to generate* 4α *sequences* $(t_{\ell,1}, \ldots, t_{\ell,s})$, *with* $1 \le \ell \le 4\alpha$, *of elements of* S, *then*

$$\mathbf{Prob}\Big(\exists k, \ell \in \{1,\ldots,4\alpha\} \exists i, j \in \{1,\ldots,s\} \exists \mu \in \{1,\ldots,m\} : \\ t_{k,i} \ne t_{\ell,j} \wedge t_{k,i}, t_{\ell,j} \in S_\mu\Big) \;>\; 1 - n^{-\alpha}.$$

Proof. By Corollary B.3, for each fixed ℓ the probability that the sequence $t_{\ell,1}, \ldots, t_{\ell,s}$ hits two different elements in the same subset S_μ is at least $1 - 4\sqrt{n}/s \ge 1 - n^{-1/4}$. By independence, the probability that this happens for one of the 4α sequences is at least $1 - (n^{-1/4})^{4\alpha}$; clearly, this is also a lower bound on the probability that the whole sequence $t_{\ell,i}$, with $1 \le \ell \le 4\alpha$ and $1 \le i \le s$, hits two different elements in the same set S_μ. ∎

Lemma B.5 *Let* $S = \{1, \ldots, n\}$ *for some* $n \ge 800$ *and take* $s = 4\lceil n^{3/4} \rceil$. *Then the random experiment described in Corollary B.3 can be carried out in* $o(n)$ *time using a sample space of size* $O(n^6)$ *(or, informally, using* $6\log_2 n + O(1)$ *random bits).*

Proof. Let us assume for the time being that a prime number p with $s < p \le 2s$ is given. (We will see at the end of the proof how such a p can be found within the time bound claimed.) According to [9], a 4-independent sequence $t'_1, \ldots, t'_p$, where each t'_j is uniformly distributed in $\{0, \ldots, p-1\}$, can be generated as follows: Choose 4 coefficients $\gamma'_0, \gamma'_1, \gamma'_2, \gamma'_3$ randomly from $\{0, \ldots, p-1\}$ and let

$$t'_j = \left(\sum_{r=0}^{3} \gamma'_r \cdot j^r\right) \bmod p, \quad \text{for } 1 \le j \le p.$$

By repeating this experiment once (independently), we obtain another such sequence $t''_1, \ldots, t''_p$. We let

$$t_j = 1 + (t'_j + p t''_j) \bmod n, \quad \text{for } 1 \le j \le s.$$

Clearly, the overall size of the sample space is $(p^4)^2 = p^8 = O(n^6)$, and the time needed for generating the sample is $O(s)$. We must show that the distribution of $t_1, \ldots, t_s$ satisfies conditions (a) and (b) of Corollary B.3. Since the two sequences $(t'_p, \ldots, t'_p)$ and $(t''_p, \ldots, t''_p)$ originate from independent experiments and each of them is 4-independent, the sequence

$$t'_1 + pt''_1, \ldots, t'_s + pt''_s$$

is 4-independent; hence the same is true for $t_1, \ldots, t_s$, and (a) is proved. Further, $t'_j + pt''_j$ is uniformly distributed in $\{0, \ldots, p^2 - 1\}$, for $1 \le j \le s$. From this, it is easily seen that, for $x \in S$,

$$\mathbf{Prob}(t_j = x) \in \left\{ \left\lfloor \frac{p^2}{n} \right\rfloor \cdot \frac{1}{p^2}, \left\lceil \frac{p^2}{n} \right\rceil \cdot \frac{1}{p^2} \right\}.$$

Now observe that $\lfloor p^2/n \rfloor / p^2 < 1/n < \lceil p^2/n \rceil / p^2$, and that

$$\left\lceil \frac{p^2}{n} \right\rceil \cdot \frac{1}{p^2} - \left\lfloor \frac{p^2}{n} \right\rfloor \cdot \frac{1}{p^2} \le \frac{1}{p^2} < \frac{1}{s^2} \le \frac{1}{16n^{3/2}} = \frac{1}{16\sqrt{n}} \cdot \frac{1}{n} < \frac{\varepsilon}{n},$$

where we used that $n \ge 800$, whence $1/(16\sqrt{n}) < 1/400 = 0.0025 = \varepsilon$. This proves (b).

Finally, we briefly recall the fact that a prime number in the range $\{s + 1, \ldots, 2s\}$ can be found deterministically in time $O(s \log\log s)$. (Note that we should not use randomization here, as we must take care not to use too many random bits.) The straightforward implementation of the Eratosthenes sieve (see, e. g., [31, Section 3.2]) for finding all the primes in $\{1, \ldots, 2s\}$ has running time

$$O\left(s + \sum_{\substack{p \le \sqrt{2s} \\ p \text{ prime}}} \lceil 2s/p \rceil\right) = O\left(s \cdot \left(1 + \sum_{\substack{p \le \sqrt{2s} \\ p \text{ prime}}} \frac{1}{p}\right)\right) = O(s \log\log s),$$

where the last estimate results from the fact that

$$\sum_{\substack{p \le x \\ p \text{ prime}}} \frac{1}{p} = O(\log\log x).$$

(For instance, this can easily be derived from the inequality $\pi(2n) - \pi(n) < 7n/(5 \ln n)$, valid for all integers $n > 1$, which is proved in [31, Section 3.14].) ∎