# 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 §

Tietojenkäsittelyopin laitos
Joensuun yliopisto
PL 111
SF-80101 Joensuu, Finland

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

†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.

**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 probabilistic 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 above problems can be solved by probabilistic 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 $\text{LOG}_2$ and $\text{EXP}_2$ 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 $\text{LOG}_2$ and $\text{EXP}_2$ are not allowed, 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.

We consider two variants of the algorithm for the closest-pair problem. One uses only $O(\log n + \log U)$ random bits and still has probability $O(n^{-\alpha})$ of exceeding the time bound $O(n)$, where $\alpha \geq 1$ is a constant that can be chosen arbitrarily. The other one uses $O(n \log n + \log U)$ random bits, but reduces the probability that the time bound is exceeded to $2^{-n^{\Omega(1)}}$.

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 $\text{LOG}_2$ and $\text{EXP}_2$ 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.

# 1 Introduction

The *closest-pair problem* is often introduced as the first nontrivial proximity problem in computational geometry — see, e. g., [21]. 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 [22] proposed an algorithm for solving the closest-pair problem. The key idea of the algorithm is to determine the minimal distance $\delta_0$ within a random sample of points. When the points are grouped according to a grid with resolution $\delta_0$, 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 $\delta_0$-grid. Rabin suggested that this could be implemented by dividing the coordinates of the points by $\delta_0$, truncating the quotients to integers, and hashing the resulting integer $d$-tuples. Fortune and Hopcroft [12], 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 $\delta_0$-grid. The indices are integers in the range $\{1, \ldots, n\}$, and distinct cells have distinct indices. Recently, two other (simple) closest-pair algorithms with linear expected running time were proposed by Golin et al. [13] and Khuller and Matias [15]. Faced with a similar grouping problem, both papers refer to a randomized hashing procedure without specifying the details.

On a real RAM (for the definition see [21]), where the generation of random numbers, comparisons, arithmetic operations from $\{+, -, *, /, \sqrt{\ }\}$, and FINDBUCKET require unit time, the random-sampling algorithm of Rabin runs in $O(n)$ expected time [22]. (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 [12].) Moreover, Rabin's algorithm is extremely reliable: the probability that the running time exceeds its expected value by more than a constant factor is exponentially small, i. e., of the form $2^{-n^{\Omega(1)}}$. (We call a probabilistic algorithm that always returns a correct answer *reliable* if the probability that the algorithm exceeds its expected running time by more than a constant factor tends to 0 as the input size grows to infinity.) Golin et al. present a variant of their algorithm that is reliable but has running time $O(n \log n / \log\log n)$.

The above time bounds should be contrasted with the fact that in the algebraic computation tree model (where the operations allowed are comparisons and arithmetic operations from $\{+, -, *, /, \sqrt{\ }\}$, but where indirect addressing is not modelled), $\Theta(n \log n)$ is known to be the complexity of the closest-pair problem.

Algorithms proving the upper bound were provided by, for example, Bentley and Shamos [6] and Schwarz et al. [24]. The lower bound follows from the corresponding lower bound derived for the element-distinctness problem by Ben-Or [5]. The $\Omega(n \log n)$ lower bound is valid even if the coordinates of the points are integers [26] 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 with high probability. Second, we modify Rabin's algorithm so that only very few random bits are needed, but still a reasonable 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 allowed 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 some integer key from the set $\{0, \ldots, U-1\}$, 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 $1 \leq i < j \leq n$ and $A[i]$ and $A[j]$ have the same key, then $A[k]$ has the same key for all $k$ with $i \leq k \leq j$. Note that full sorting is not necessary, since no order is prescribed for items with different keys. 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 from $\{0, \ldots, U-1\}$. It will be obvious that our algorithms can be extended to solve the general duplicate-grouping problem.

For solving the duplicate-grouping problem, one can employ perfect-hashing schemes in the style of [11], which work in linear expected time. Actually, the original procedure from [11] is not suitable, the reason being that it is not able to deal properly with repeated values, which may occur when the hashing scheme is used as a subroutine in Rabin's algorithm (or in those of [13] or [15]). One solution would be to use the dynamic perfect-hashing scheme of [10], which is able to accommodate duplicates. However, this algorithm does not offer any guarantees on the running time beyond the fact that its expectation is linear. As the duplicate-grouping algorithm is to be used as a subroutine in the closest-pair algorithm, which we want to be reliable, we provide an alternative solution. Assuming that $U$ is a power of 2 given as part of the input, the problem is shown to be solvable in $O(n)$ time with high probability using arithmetic operations from $\{+, -, *, \text{DIV}\}$ only. 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 $\text{LOG}_2$ and $\text{EXP}_2$ among the unit-time operations. It is essential to

[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".

note that our algorithms for duplicate grouping are *conservative* in the sense of [16], i. e., all numbers manipulated during the computation have $O(\log n + \log U)$ bits.

Technically as an ingredient of the duplicate-grouping algorithm, we introduce a new universal class of hash functions — more precisely, we prove that the class of all multiplicative hash functions [17] is universal in the sense of [7]. This class seems quite attractive, since the functions in the class can be evaluated very efficiently (only multiplications and shifts of binary representations are needed). Actually, we know of no other universal class whose functions can be evaluated more cheaply.

On the basis of the duplicate-grouping algorithm 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 $\{+, -, *, \text{DIV}\}$ 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 $a \,\text{DIV}\, b = \lfloor a/b \rfloor$, 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, two algorithms for the duplicate-grouping problem are presented. The 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 hash functions, the problem of generating primes, and probability estimates are given in the three parts of an appendix.

# 2 Randomized duplicate grouping

In this section we present two randomized algorithms for solving the duplicate-grouping problem. As technical tools, we discuss a simple universal class of hash functions and a method for generating numbers that are prime with high probability.

## 2.1 A deterministic algorithm for duplicate grouping

For a specification of the duplicate-grouping problem see the introduction. Let $S = \{x_1, x_2, \ldots, x_n\}$ be the multiset to be grouped, containing $n$ integers from the set $\{0, \ldots, U-1\}$. When $O(n+U)$ space is available, the duplicate grouping is easily done in $O(n)$ time. E. g., we can use the following naive algorithm, similar to one phase of radix sort (see [2]): Assume that the input numbers are stored in an array $S[1..n]$. Let $L[0..U-1]$ be an array 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 $x_i = S[i]$ is inserted in the list with header $L[S[i]]$. During the third scan, the groups are output: For $i = 1, \ldots, n$, if the list with header $L[S[i]]$ is nonempty, this list is written to consecutive positions of the output array and $L[S[i]]$ is made to point to an empty list again.

Unfortunately, the naive duplicate-grouping algorithm wastes space. Space efficiency can be achieved by compressing the work area by means of hashing: using a randomized hashing scheme, the duplicate-grouping problem can be solved in linear time and space with high probability.

## 2.2 Duplicate grouping via multiplicative universal hashing

Here we demonstrate how to perform duplicate grouping by means of a simple application of universal hashing, as introduced by Carter and Wegman [7]. We first assume that $U$ is a known power of 2, say $U = 2^k$ (the opposite case is discussed at the end of Subsection 2.3).

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

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

The class $\mathcal{H}_{k,\ell}$ contains $2^{k-1}$ hash functions (it is not hard to see that they are all distinct). The following is proved in Section A of the appendix.

**Lemma 2.1.** *Let $k$ and $\ell$ be integers with $1 \le \ell \le k$. If $x, y \in \{0, \ldots, 2^k - 1\}$ are distinct and $h_a \in \mathcal{H}_{k,\ell}$ is chosen at random, then*

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

**Remark 2.2.** *The lemma says that the class $\mathcal{H}_{k,\ell}$ consisting of all "multiplicative hash functions" is 2-universal in the sense of [20, p. 140] (this notion slightly generalizes that of [7]). As discussed in [17, p. 509] ("the multiplicative hash method"), 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 [11, 7], arithmetic in finite fields [7], matrix multiplication [7], or convolution of binary strings over the two-element field [18], 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 [11] and [10], in place of the functions of the type $x \mapsto ax \bmod p$, for a prime $p$, which were used in these papers, and which involve integer division.*

The following is a well-known property of universal classes (see, e. g., [11]).

**Lemma 2.3.** *Let $n$, $k$ and $\ell$ be positive integers with $\ell \leq k$ 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*

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

*Proof.* By Lemma 2.1,

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

■

**Theorem 2.4.** *Let $U \geq 2$ be a known power of 2 and let $\alpha \geq 1$ be an arbitrary integer. The duplicate-grouping problem for a multiset $S$ of $n$ integers in the range $\{0, \ldots, U-1\}$ can be solved by a (conservative) probabilistic 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 $\log U$ random bits.*

*Proof.* Let $k = \log U$ and $\ell = \lceil (\alpha + 2) \log n \rceil$ (unsubscripted "log" always denotes the logarithm to base 2) and assume without loss of generality that $1 \leq \ell \leq k$. It is easy to compute $2^\ell$ in $O(\alpha \log n)$ time. The elements of $S$ are 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 $\{0, \ldots, 2^\ell - 1\}$. (Since the MOD operation can be expressed in terms of $-$, $*$, and DIV, the hash function can be evaluated in $O(1)$ time.) Third, the resulting pairs $(x, h(x))$, where $x \in S$, are sorted by radix sort according to their second components. (Since the values of $h$ are integers of $\lceil (\alpha + 2) \log n \rceil$ bits each, they can be sorted by radix sort with $n$ buckets in $\alpha + 3$ rounds, that is, in $O(\alpha n)$ time — see, e. g., [2, p. 77 ff.].) Fourth, it is checked if all elements of $S$ that have the same hash value are in fact equal. (This is easily done in $O(n)$ time. In case the check indicates that the algorithm has failed, one can get a correct output by sorting in $O(n \log n)$ time, without impairing the linear expected running time.)

The total running time of the algorithm is $O(\alpha n)$. The space requirements are dominated by those of radix sort, which needs $O(n)$ space. The result obtained

after radix sorting 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 the previous lemma. It is immediate that the algorithm is conservative and that the number of random bits needed is $k - 1 < \log U$. ∎

## 2.3 Randomized 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 a randomized perfect-hashing scheme introduced by Bast and Hagerup [3].

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 [3] a parallel algorithm for the perfect-hashing problem is described; we need the following sequential version.

**Fact 2.5** ([3])**.** *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 probabilistic algorithm that requires space $O(n)$ 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 number larger than a given number $m$. In order to find such a prime, we again use a probabilistic algorithm. The simple idea is to combine random sampling with a probabilistic primality test as given, e. g., in [23]. Such algorithms with expected running time polylogarithmic in $m$ have been described or discussed in several papers, e. g., in [4], [19], and [9]. 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 criteria. The proofs of the following two lemmas, which include the description of the algorithms, can be found in Section B of the appendix.

**Lemma 2.6.** *There is a probabilistic 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 $1/m$.*

By varying this algorithm, we obtain the following extremely reliable version.

**Lemma 2.7.** *There is a probabilistic algorithm that, for any given positive integers $m$ and $n$ with $2 \leq m \leq 2^{\lceil n^{1/4} \rceil}$, returns a number $p$ with $m < p \leq 2m$ 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 2.8.** *The algorithms of Lemmas 2.6 and 2.7 run 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 2.9.** *Let* $U \geq 2$ *be a known power of* 2. *The duplicate-grouping problem for a multiset* $S$ *of* $n$ *integers in the range* $\{0, \ldots, U-1\}$ *can be solved by a (conservative) probabilistic 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 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 2.7 to find a prime $p$ between $U$ and $2U$. Then, the hash function from Fact 2.5 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 the naive algorithm introduced at the beginning of this section. In case $U$ is large, we first "collapse the universe" by mapping the elements of $S \subseteq \{0, \ldots, U-1\}$ one-to-one into the range $\{0, \ldots, U'-1\}$ by a randomly chosen multiplicative hash function, as described in Subsection 2.2. Then, using the "collapsed" keys, we proceed as above for a universe that is not large.

It remains to analyze the time requirements and the failure probability 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 2.7 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 randomly choose a function $h$ from $\mathcal{H}_{k,\ell}$, where $U = 2^k$ is known and $\ell = \lceil n^{1/4} \rceil$. Clearly, $2^\ell$ can easily be obtained in time $O(\ell) = O(n^{1/4})$. The values $h(x)$, for $x \in S$, can be computed in time $O(|S|) = O(n)$; according to Lemma 2.3 $h$ 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 final naive grouping runs in deterministic 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 reliable since all the subroutines used are reliable. The hashing scheme of Bast and Hagerup is conservative. The justification that the other parts of the algorithm are conservative is straightforward. ∎

**Remark 2.10.** *Theorem 2.9 is theoretically stronger than Theorem 2.4, but the program based on the former result will be much more complicated. Moreover,* $n$ *must be very large before the algorithm of Theorem 2.9 is actually significantly more reliable than that of Theorem 2.4.*

In Theorems 2.4 and 2.9 we assumed that $U$ is a known power of 2. 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 $\text{LOG}_2$ and $\text{EXP}_2$ among the unit-time

operations and to use them to compute $2^{\lceil \log U \rceil}$. As soon as the required power of 2 is available, the algorithms described above can be used. Thus, Theorem 2.9 can be extended as follows (the same holds for Theorem 2.4, but only with an inverse-polynomial error probability).

**Theorem 2.11.** *The duplicate-grouping problem for a multiset of $n$ integers in the range $\{0, \ldots, U-1\}$ can be solved by a (conservative) probabilistic 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 $2^{-n^{\Omega(1)}}$.* ∎

## 2.4 Duplicate grouping for $d$-tuples

In the context of the closest-pair problem, the duplicate-grouping problem arises not for sets of integers from $\{0, \ldots, U-1\}$, but for sets of $d$-tuples of integers from $\{0, \ldots, U-1\}$, where $d$ is the dimension of the space under consideration. Even if we drop the assumption that $d$ is constant, our algorithms are easily adapted to this situation with 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 $d \log U$ bits, which may be undesirable. We sketch 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 2.12.** *Theorems 2.4 and 2.9 (and 2.11) 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.* (Sketch.) We indicate how the algorithms described in the proofs of Theorems 2.4 and 2.9 have to be changed in order to accommodate $d$-tuples. Assume an array $S$ containing $n$ $d$-tuples of numbers in $\{0, \ldots, U-1\}$ is given as input. We treat the components $d' = 1, \ldots, d$ one after the other. Let $1 \le d' \le d$ and assume inductively that the tuples are already grouped with respect to components $1, \ldots, d'-1$. Now choose a hash function for the up to $n$ numbers occurring in component $d'$ of the tuples. (In the case of Theorem 2.9, choose a perfect hash function with range $\{0, \ldots, cn\}$ for this set. In the situation of Theorem 2.4 choose a multiplicative hash function for this set at random. Actually, in the latter case the same function should be used for all $d'$, with $1 \le d' \le d$, in order to avoid using more than $\log U$ random bits.) Now, in each of the segments of $S$ in which components $1, \ldots, d'-1$ are constant, radix sort with respect to the hash value of

component $d'$ is applied, using $n$ buckets. This last step needs space $O(n)$, since the same scratch space for the bucket lists needed in radix sort can be used for all segments, and it takes time $O(n)$, since radix sort in each of the segments takes linear time. ∎

# 3 A randomized closest-pair algorithm

In this section we describe a variant of the random-sampling algorithm of Rabin [22] 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 in the algorithm (and which is easily generalized to higher dimensions). For all $\delta > 0$, a *grid $G$ with resolution $\delta$*, or briefly a *$\delta$-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 $\delta$. 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 $\delta$).

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

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 elements of $\{1, \ldots, n\}$ randomly. Let $T := \{t[1], \ldots, t[s]\}$ and take $s' := |T|$. Store the elements $p_t$ with $t \in T$ in an array $B[1..s']$ ($B$ may contain duplicates if $S$ does).

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

3. Check whether there exists a pair of points whose distance is smaller than $\delta_0$. For this, draw a grid $G$ with resolution $\delta_0$ 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 falling into the same cell of $G_i$.

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

```
proc 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 in [1..n]
% Eliminate repetitions in t[1..s]; store the chosen elements in an array B.
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; B[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(B[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 := j + 1, j + 1
         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_a, p_b) > dist(p_c, p_d) then p_a, p_b := p_c, p_d
return (p_a, p_b)
```

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

In contrast to Rabin's algorithm [22], 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 [22], 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 $S[i]$ 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

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

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 and compute the maximum and minimum coordinates.

The correctness of the procedure "randomized-closest-pair" follows from the fact that, since $\delta_0$ is an upper bound on the minimum distance between any two points of the set $S$, a closest pair falls into the same cell in at least one of the shifted $2\delta_0$-grids. Note that the algorithm works correctly even in the case that sampling or grouping fails. The only harm caused by a failure in the grouping procedure is that points falling into different cells are grouped together. However, *all* the points falling into the same cell are still grouped together, so that a closest pair will be found. Therefore a failure in sampling or grouping can only increase the running time. However, we will see in the next section that the running time is linear with high probability.

**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 $\delta_0$ 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 $\delta_0$ 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 below.) The square root operation can also be avoided in the deterministic algorithm as shown, for example, in [24]. The tricks just mentioned suffice for showing that the closest-pair algorithm can be made to work for an arbitrary fixed Minkowski metric $L_p$, without computing pth roots.*

**Remark 3.2.** *The randomized closest-pair algorithm generalizes naturally to any $d$-dimensional space. Note that while two shifts (by 0 and $\delta_0$) 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" 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 successive points. In the* two-dimensional *case one can use the simple plane-sweep algorithm of Hinrichs et al. [14]. In the* multi-dimensional *case, the divide-and-conquer algorithm by Bentley and Shamos [6] and the incremental algorithm by Schwarz et al. [24] 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 in detail only the case of two-dimensional problems. 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 (cf. 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 $\alpha$, 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 [22] as well as in those of Golin et al. [13] and Khuller and Matias [15] it was assumed that all input points are distinct, and the problem of duplicate points in the input was not addressed at all.

Varying a notion from [22], 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 $S_\mu$ 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 $N(S) \geq n$. Then, by Corollary C.2 in Appendix C, 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 [22], expresses the work done in Step 3b when the subproblems are solved by the brute-force algorithm.

**Lemma 4.1** ([22])**.** *Let $S$ be a multiset of $n$ points in the plane. Further, let $G$ be a grid with resolution $\delta$, and let $G'$ be one of the four grids with resolution $2\delta$ 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

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

This implies

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

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

**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* $N(S, G') \leq 2^d N(S, G) + \frac{1}{2}(2^d - 1)n$.

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

*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 $N(S, G') \geq n$ 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 $\delta$. Further, let $G'$ be an arbitrary grid with resolution at most $\delta$. Then $N(S, G') \leq 16N(S, G) + 6n$.*

*Proof.* Let $G_i$, for $i = 1, 2, 3, 4$, be the four different grids with resolution $2\delta$ 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 $N(S, G') \leq \sum_{i=1}^{4} N(S, G_i)$. Applying Lemma 4.1 and summing up yields $N(S, G') \leq 16N(S, G) + 6n$, 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 C.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 time $O(n)$.

Since the real number $\delta_0$ calculated by the algorithm in Step 2 is bounded by the distance of two points in the same cell of $G^*$, we must have $\delta_0 \leq 2\delta^*$. (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 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 as large as $2\delta_0$. 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 2.11, 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 probabilistic 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 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 $[0,\lceil\delta_{\max}/\delta_{\min}\rceil]$. By Theorem 2.12, parts (1) and (2) of the theorem follow directly from the corresponding parts of Theorem 2.11. 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. ∎

**Corollary 4.6.** *The closest-pair problem for a multiset of $n$ points in $d$-dimensional space with integer coordinates in the range $\{0,\ldots,U-1\}$, where $d \geq 1$ is a fixed integer, can be solved by a conservative probabilistic 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 $2^{-n^{\Omega(1)}}$.*

*Proof.* Parts (1) and (2) follow from the corresponding parts of Theorem 4.5 and the fact that $\delta_{\max}/\delta_{\min} = O(U)$. It is clear that the randomized closest-pair algorithm is conservative, since all its subroutines are conservative. ∎

Even if the number of random bits used is severely restricted, we can still retain an algorithm that is quite reliable.

**Theorem 4.7.** *Let $\alpha, d \geq 1$ be arbitrary integers. The closest-pair problem for a multiset of $n$ points in $d$-dimensional space can be solved by probabilistic algorithms with the time and space requirements stated in Theorem 4.5 and Corollary 4.6 that use only $O(\alpha\log n+\log(\delta_{\max}/\delta_{\min}))$ or $O(\alpha\log n+\log U)$ random bits, respectively, and exceed 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 $4\alpha$ independently chosen sequences of 4-independent random values that are approximately uniformly distributed in $[1..n]$. This random experiment and its properties are described in detail in Corollary C.4 and Lemma C.5 in Section C of the appendix. The time needed is $o(n)$, and the number of random bits needed is $O(\alpha \log n)$. The duplicate grouping is performed with the simple method described in Section 2.2. This requires only $O(\log U)$ or $O(\log(\delta_{\max}/\delta_{\min}))$ random bits, respectively. The analysis is exactly the same as in the proof of Theorem 4.5, except that Corollary C.4 is used instead of Corollary C.2. ∎

**Remark 4.8.** *The worst-case complexity of our closest-pair algorithm is easily improved to $O(n \log n)$, or $O(n \log n + \log\log(\delta_{\max}/\delta_{\min}))$, by solving the small subproblems (in Step 3b) with an optimal deterministic algorithm (as in Step 2). This is because $\sum_{i=1}^{m} n_i \log n_i \leq n \log n$ if $\sum_{i=1}^{m} n_i = n$, and because we never use more than linear time in sampling or grouping (that is, in Steps 1, 2, and 3a).*

# 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 $\text{LOG}_2$ and $\text{EXP}_2$, 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 [21]), can be solved as efficiently as the one-dimensional closest-pair problem on the real RAM.

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

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

In the light of these results, one should not attach exaggerated weight to a proof showing some problem to be at least as hard as, say, the element-distinctness problem.

Finally, we would like to mention practical experiments with our algorithms. The experiments were conducted by Tomi Pasanen (University of Turku, Finland). He found that the duplicate-grouping algorithm based on radix sort (with $\alpha = 3$), i.e., the algorithm described in Theorem 2.4, behaved 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 2.4) 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 is utilized.

# Acknowledgements

We would like to thank Ivan Damgård for his comments concerning Lemma 2.6 and Tomi Pasanen for his assistance in evaluating the practical efficiency of the algorithms. 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, 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, J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[3] H. Bast, T. Hagerup, Fast and reliable parallel hashing, in *Proc. of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures* (1991), pp. 50–61.

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

[5] M. Ben-Or, Lower bounds for algebraic computation trees, in *Proc. of the 15th Annual ACM Symposium on Theory of Computing* (1983), pp. 80–86.

[6] J. L. Bentley, M. I. Shamos, Divide-and-conquer in multidimensional space, in *Proc. of the 8th Annual ACM Symposium on Theory of Computing* (1976), pp. 220–230.

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

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

[9] I. Damgård, P. Landrock, C. Pomerance, Average case error estimates for the strong probable prime test, Submitted to *Math. Comp.*.

[10] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, R. E. Tarjan, Dynamic perfect hashing: Upper and lower bounds. Technical Report 77, Universität-Gesamthochschule Paderborn, Fachbereich Mathematik/Informatik, Paderborn, Germany, 1991. To appear in *SIAM J. Comput.* A preliminary version appeared in *Proc. of the 29th IEEE Symposium on Foundations of Computer Science* (1988), pp. 524–531.

[11] M. L. Fredman, J. Komlós, E. Szemerédi, Storing a sparse table with $O(1)$ worst case access time, *J. Amer. Math. Soc.* **31** (1984) 538–544.

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

[13] M. Golin, R. Raman, C. Schwarz, M. Smid, Simple randomized algorithms for closest pair problems, Technical Report MPI-I-92-155, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1992.

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

[15] S. Khuller, Y. Matias, A simple randomized sieve algorithm for the closest-pair problem, in *Proc. of the 3rd Canadian Conference on Computational Geometry* (1991), pp. 130–134.

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

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

[18] Y. Mansour, N. Nisan, P. Tiwari, The computational complexity of universal hashing, in *Proc. of the 22nd Annual ACM Symposium on Theory of Computing* (1990), pp. 235–243.

[19] Y. Matias, U. Vishkin, On parallel hashing and integer sorting, Technical Report UMIACS–TR–90–13.1, University of Maryland, College Park, MD, 1990. Journal version: *J. Algorithms* **12** (1991) 573–606.

[20] K. Mehlhorn, *Data Structures and Algorithms, Vol. 1: Sorting and Searching*, Springer-Verlag, 1984.

[21] F. P. Preparata, M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, 1985.

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

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

[24] C. Schwarz, M. Smid, J. Snoeyink, An optimal algorithm for the on-line closest-pair problem, in *Proc. of the 8th Annual Symposium on Computational Geometry* (1992), pp. 330–336.

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

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

# A Multiplicative universal hashing

We prove the following lemma from Section 2.

**Lemma 2.1.** *Let $k$ and $\ell$ be integers with $1 \le \ell \le k$. If $x, y \in \{0, \ldots, 2^k - 1\}$ are distinct and $h_a \in \mathcal{H}_{k,\ell}$ is chosen at random, then*

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

*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

$$az \bmod 2^k \in \{1, \ldots, 2^{k-\ell} - 1\} \cup \{2^k - 2^{k-\ell} + 1, \ldots, 2^k - 1\}. \qquad \text{(A.1)}$$

In order to estimate the number of $a \in A$ that satisfy (A.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 (A.1) is the same as the number of $a \in A$ that satisfy

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

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 (A.2). For smaller $s$, the number of $a \in A$ satisfying (A.2) is at most $2^{k-\ell}$. Hence the probability that a randomly chosen $a \in A$ satisfies (A.1) is at most $2^{k-\ell}/2^{k-1} = 1/2^{\ell-1}$. ∎

# B Generating primes

In this section, we provide proofs of Lemmas 2.6 and 2.7.

**Lemma 2.6.** *There is a probabilistic 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 probabilistic primality test of Rabin [23] (see also [9]). 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 well known that the function $\pi(x) := |\{p|\, p \le x \text{ and } p \text{ is prime}\}|$ satisfies

$$\pi(2x) - \pi(x) \ge \frac{x}{3\ln(2x)}, \quad \text{for } x \ge 1.$$

(For a complete proof of this fact, also known as "Finsler's inequality", see [25, Sects. 3.10 and 3.14].) Hence, for all $m \ge 1$ 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

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

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 B.1.** *The problem of generating primes is discussed by Damgård et al. [9] in greater detail. Their analysis shows that the proof of Lemma 2.6 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 2.7.** *There is a probabilistic 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 from Lemma 2.6 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

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

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

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

■

# C 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. In principle, the lemma was proved in [22, Lemma 6]. In Subsection C.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 Subsection C.2.

## C.1 The sampling lemma

**Lemma C.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 $s \geq 8\sqrt{n}$,*

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

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

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

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 (C.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 (C.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:

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

(For another, though simpler, application of Chebyshev's inequality in a similar context see [8]).

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

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

Thus,

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

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

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

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

$$\mathbf{cov}(X_{i,j}^{\pi}, X_{i',j'}^{\pi'}) = \mathbf{E}(X_{i,j}^{\pi} \cdot X_{i',j'}^{\pi'}) - \mathbf{E}(X_{i,j}^{\pi}) \cdot \mathbf{E}(X_{i',j'}^{\pi'})$$

we may write

$$\mathbf{Var}(X) = \mathbf{E}(X^2) - (\mathbf{E}(X))^2 = \sum_{\pi, \pi' \in P(D)} \sum_{\substack{1 \le i < j \le s \\ 1 \le i' < j' \le s}} \mathbf{cov}(X_{i,j}^{\pi}, X_{i',j'}^{\pi'}). \tag{C.7}$$

We split the summands $\mathbf{cov}(X_{i,j}^{\pi}, X_{i',j'}^{\pi'})$ 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

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

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

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

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

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

*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 $\mathbf{cov}(X^{\pi}_{i,j}, X^{\pi'}_{i',j'}) \leq 0$.

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.* $|J| = 3$. 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 $\pi$, while the two remaining elements of $J$ are both mapped to the other element of $\pi$, 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 $s^3 \cdot N(D) \cdot 2/n^3 \leq 2.2s^3/n^2$.

*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 $\pi$ (and, implicitly, $\pi'$) 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 $\pi$ and the remaining element of $\{i',j'\}$ is mapped to the remaining element of $\pi'$, 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

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

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

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

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

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

By (C.3) we have

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

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

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

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 C.2.** *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 $s \geq 9\sqrt{n}$,*

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

*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 C.1, in each of the corresponding subexperiments the probability that two elements in the same subset $S_\mu$ 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 $S_\mu$. ∎

## C.2 Sampling with few random bits

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

**Corollary C.3.** *Let $n$, $m$, $s$, $S$, and $D$ be as in Lemma C.1. Then the conclusion of Lemma C.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 $\{i, j, k, l\} \subseteq \{1, \ldots, s\}$ 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 C.1. We indicate the slight changes that have to be made. Equation (C.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 (C.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 (C.6) remains valid. The contributions to $\mathbf{Var}(X)$ of the summands of the various classes defined in the proof of Lemma C.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:* $\leq 0$.

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

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

Finally, estimate (C.9) is replaced by

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

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

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

**Corollary C.4.** *In the situation of Lemma C.1, let $s \geq 4\lceil n^{3/4} \rceil$, and let $\alpha \geq 1$ be an arbitrary integer. If the experiment described in Corollary C.3 is repeated independently $4\alpha$ times to generate $4\alpha$ sequences $(t_{l1}, \ldots, t_{ls})$, with $1 \leq l \leq 4\alpha$, of elements of $S$, then*

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

*Proof.* By Corollary C.3, for each fixed $l$ the probability that the sequence $t_{l1}, \ldots, t_{ls}$ hits two different elements in the same subset $S_\mu$ is at least $1 - 4\sqrt{n}/s \geq 1 - n^{-1/4}$. By independence, the probability that this happens for one of the $4\alpha$ sequences is at least $1 - (n^{-1/4})^{4\alpha}$; clearly, this is also a lower bound on the probability that the whole double sequence $t_{li}$, with $1 \leq l \leq 4\alpha$ and $1 \leq i \leq s$, hits two different elements in the same set $S_\mu$. ∎

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

*Proof.* Let us assume for the time being that a prime number $p$ with $s < p \leq 2s$ 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 [8], 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 randomly 4 coefficients $\gamma'_0, \gamma'_1, \gamma'_2, \gamma'_3$ from $\{0, \ldots, p-1\}$ and let

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

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

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

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 C.3. Since the two sequences $(t'_p, \ldots, t'_p)$ and $(t''_p, \ldots, t''_p)$ originate from independent experiments, and each one of them is 4-independent, the sequence

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

is 4-independent; hence the same is true for $t_1, \ldots, t_s$, and (a) is proved. Further, $t'_j + p t''_j$ is uniformly distributed in $\{0, \ldots, p^2 - 1\}$, for $1 \leq j \leq s$. 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

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

where we used that $n \geq 800$, 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., [25, Sect. 3.2]) for finding all the primes in $\{1, \ldots, 2s\}$ has running time

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

where the last estimate results from the well-known fact that

$$\sum_{\substack{p \le y \\ p \text{ prime}}} \frac{1}{p} = O(\log \log y), \qquad \text{for } y \ge 1.$$

(E. g., this can easily be derived from the inequality $\pi(2n) - \pi(n) < 7n/(5 \ln n)$, for $n > 1$, which is proved in [25, Sect. 3.14].) ∎