# A Simple Randomized Sieve Algorithm for the Closest-Pair Problem

*Samir Khuller* *
Dept. of Computer Science
University of Maryland
College Park, MD 20742

*Yossi Matias* †
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974

**Abstract**

We present a linear time randomized sieve algorithm for the closest-pair problem. The algorithm as well as its analysis are simple. The algorithm is extended to obtain a randomized linear time approximation algorithm for the closest bichromatic pair problem.

*Part of this work was done while this author was with University of Maryland Institute for Advanced Computer Studies, and partially supported by NSF grants CCR-8906949, CCR-9111348 and CCR-9103135.

†Part of this work was done while this author was with University of Maryland Institute for Advanced Computer Studies and with the Computer Science Dept., Tel Aviv University, and partially supported by NSF grants CCR-8906949 and CCR-9111348.

# 1. Introduction

The closest-pair problem can be found in almost any algorithms text-book as a basic problem in computational geometry (see, e.g., [CLR90, Man89, PS86]). Deterministic algorithms that run in $O(n \log n)$ time are due to [Ben80, BS76, HNS88, SH75]. These algorithms are optimal in the algebraic decision-tree model of computation, where a matching lower bound of $\Omega(n \log n)$, even for the 1-dimensional closest-pair problem, is implied by a lower bound for element distinctness [Ben83, Yao91]. The lower bound only holds when the floor function is not allowed, as was shown by Fortune and Hopcroft [FH79], who obtained an $O(n \log\log n)$ deterministic algorithm by making use of the floor function. This simply stated problem was used by Rabin in a classic paper [Rab76], to illustrate the power of *randomization*, where he gave an algorithm that takes only $O(n)$ expected running time. Rabin uses a random sampling technique to decompose the problem into "small" subproblems for which the total cost of a brute force method is expected to be linear, using the floor function. His algorithm, although simple, has a somewhat complicated analysis.

In this paper, we present a novel approach using a sieve technique that yields a simple new algorithm. The algorithm takes linear expected time, using the floor function, and has a simple analysis

The *closest-pair* problem is defined as follows: given a set $S$ of $n$ points in $\Re^d$, for some constant $d > 0$, the task is to find a closest pair of points (in Euclidean distance). The algorithm is described for the plane and can be easily extended to run in linear expected time for any fixed dimension.

A generalization of the closest-pair problem is the following *closest bichromatic pair* problem: Given a set of $n$ colored points in $\Re^d$, for some constant $d > 0$, find a closest pair of points that are differently colored ("bichromatic pair"); i.e., find a point $p$ and a point $q$ that have different colors, such that the distance between $p$ and $q$ is minimum among all the bichromatic pairs. An instance of the problem where each point is colored by one of *two* colors was considered by Agarwal et al. [AESW90]. For an input consisting of $m$ red points and $n$ blue points from $\Re^3$ (i.e., for $d = 3$), they give a randomized algorithm running in expected time $O((km \log k \log m)^{2/3} + m \log^2 k + k \log^2 m)$. (This has applications in solving the Euclidean minimum spanning tree problem as was shown by [AESW90].)

We extend our closest-pair sieve algorithm and obtain an *approximation* algorithm for the closest bichromatic pair problem. Our algorithm takes $O(n)$ expected time for any fixed $d$ and for any number of colors, and finds a bichromatic pair whose distance is no more than $(1 + \epsilon)$ times the distance of the actual closest bichromatic pair, for any fixed $\epsilon$.

The algorithm is described for the plane and can be easily extended to run in linear expected time for any fixed dimension.

The more general *all nearest neighbors* problem, in which we are required to compute the closest neighbors for each point in the set $S$, has also been well studied [Ben80]. A randomized $O(n \log n)$ algorithm was given by [Cla83], and a deterministic $O(n \log n)$ algorithm was given by

[Vai89]. In the last few years, the dynamic and semi-dynamic ("on-line") versions of the closest pair problem were extensively studied. See [SSS92, Mat93] and references given therein for work on these problems.

We assume that we can compute the distance between two points in constant time; i.e., square-roots can be computed in constant time. We also assume that the floor operation can be computed in constant time (this assumption is implicit in Rabin's paper as well).

## 2. The Sieve Closest-Pair Algorithm

Let $S$ be the initial set of given points. We use $\delta(S)$ to denote the distance between points of a closest pair in $S$. We will assume that the points are not numbered, and we will number them as the algorithm proceeds. The algorithm consists of two stages. We first compute an approximation for $\delta(S)$; then the approximation is used to compute $\delta(S)$. The main idea of the algorithm is to do a simple "filtering" process, in which points are deleted from the set. Let $S_i$ be the set of remaining points at the start of iteration $i$ (initially $S_1$ is $S$). The filtering continues as long as $S_i$ is non-empty. In the filtering, the sizes of the sets $S_i$ are shrinking geometrically. At the end of the process we get an approximation (up to a factor of 3) to the closest-pair distance. We then use a simple technique to find a closest pair.

**The filtering process:** Pick at random a point from $S_i$ and call it $x_i$. The distance to the closest point from a point $x$, *in the current set* $S_i$, is defined to be $d(x)$. Compute $d(x_i)$ by computing the distance from $x_i$ to all points in $S_i$. To obtain $S_{i+1}$ we delete from $S_i$ all points $x$ such that $d(x) \geq d(x_i)$. In this process, we may also delete some points $x$ such that $d(x_i) > d(x) > d(x_i)/3$. However, all points $x$ such that $d(x) \leq d(x_i)/3$ will remain in $S_{i+1}$. We stop the filtering process when the set $S_i$ becomes empty. Let $i^*$ be the smallest index such that $S_{i^*+1} = \emptyset$ and $S_{i^*} \neq \emptyset$. Let $x_{i^*}$ be the point selected at random from $S_{i^*}$ at iteration $i^*$. The closest-pair distance $\delta(S)$ is at most $d(x_{i^*})$ and at least $d(x_{i^*})/3$. Thus at the end of the filtering process we find an approximation to $\delta(S)$ within a factor of 3.

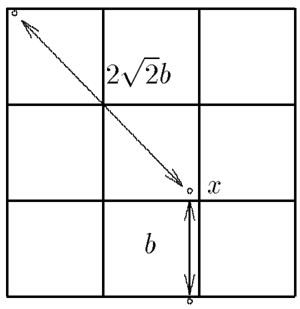


Figure 1: Neighborhood of a point $x$

It remains to show how to implement the above process, and how to use the resulting approximation $d(x_{i^*})$ to find a closest pair. These can be done easily by the notion of "neighborhood". This notion was previously used in other algorithms for the closest-pair problem and for the more general all-nearest-neighbors problem. Consider a mesh of size $b$. The *neighborhood* of a point $x$, is the cell containing $x$ plus the 8 neighboring cells (see Figure 1). Let $N(x)$ be the set of points in the neighborhood of $x$. The following facts can be easily verified.

(1) All points whose distance from $x$ is at least $3b$ (actually $2\sqrt{2}b$) are not in $N(x)$.

(2) All points whose distance from $x$ is at most $b$ are in $N(x)$.

Note that points with distance from $x$ between $b$ and $2\sqrt{2}b$ may be either in $N(x)$ or not in $N(x)$.

**Implementation of "filtering":** Let $x_i$ be a point selected at random from set $S_i$ in iteration $i$. We build a mesh of size $b = d(x_i)/3$. A point $x$ is deleted from $S_i$ if and only if it is the only point in its neighborhood (i.e., the only point in $N(x)$). This can be implemented in $O(|S_i|)$ expected number of steps and linear space by using *hashing*. Specifically, a perfect (i.e., injective) hash function can be computed in $O(|S_i|)$ time to represent the $|S_i|$ non-empty cells in $O(|S_i|)$ space, by using the algorithm of [FKS84]. Alternatively, standard hashing techniques such as coalesced hashing can be used (see, e.g., [Knu73, page 514]). The above stated complexities can be achieved for *any input* by using hash functions chosen at random from a universal class of hash functions [CW79].

**Lemma 2.1:** *When the algorithm terminates ($S_{i^*+1} = \emptyset$), then $d(x_{i^*})/3 \leq \delta(S) \leq d(x_{i^*})$.*

*Proof:* Clearly, $\delta(S) \leq d(x_{i^*})$ (by definition). By Fact (1), in iteration $i$ all points $x$ with $d(x) \geq d(x_i)$ are deleted. Therefore, the sequence $\{d(x_i)\}$ is monotonically decreasing. Let $j^*$ be the first iteration in which a point $u$ of a closest-pair is deleted from $S_{j^*}$. By Fact (2) all points $x$ with $d(x) \leq d(x_{j^*})/3$ are not deleted from $S_{j^*}$. Note that $d(u) = \delta(S)$ since both closest-pair points are in $S_{j^*}$. Therefore, $\delta(S) \geq d(x_{j^*})/3 \geq d(x_{i^*})/3$. □

**Computing $\delta(S)$ from its approximation:** We will use the following simple facts.

Consider a mesh of size $b$ and assume that $b/3 \leq \delta(S) \leq b$. Then

(3) The neighborhood of each point in $S$ contains at most a constant number of points.

(4) Each point of a closest pair is contained in the neighborhood of the other point.

We construct a mesh of size $d(x_{i^*})$ (the approximation of $\delta(S)$). For each non-empty cell we build a list of all points in this cell. This can be implemented in linear expected time and linear space by using hashing as above. Then, for each point we find the distance to its closest point in its neighborhood, if such a point exists. By Fact (3) this can be done in constant time per point, by using a brute force method (or other more efficient techniques). Fact (4) guarantees that for each point from the closest pair, the other point is in its neighborhood. Thus, the closest pair will be found by computing the minimum over the distances.

We now give the algorithm in more detail.

**The Sieve Closest-Pair Algorithm:**

*Step 0.* Initialize $S_1$ to be $S$, and $i = 1$.

*Step 1.* Pick at random a point $x_i$ from $S_i$, and compute $d(x_i)$, the distance to the closest point in $S_i$.

*Step 2.* Construct a mesh of size $b = d(x_i)/3$; Define $X_i = \{x_j \mid N(x_j) = \{x_j\}\}$;
$S_{i+1} = S_i - X_i$. ($X_i$ is a collection of points that are alone in their neighborhood.)

*Step 3.* If $S_{i+1} \neq \emptyset$ then $i = i + 1$, and goto Step 1.
Else (if $S_{i+1} = \emptyset$) let $i^* = i$. ($d(x_{i^*})/3 \leq \delta(S) \leq d(x_{i^*})$.)

*Step 4.* Construct a mesh of size $d(x_{i^*})$. For each point $x$ in $S$, compute the distance to the closest point to $x$ in $N(x)$ (if there is such a point). Find a closest pair by computing the minimum over all computed distances.

**Analysis:**

The only thing to show is that the filtering process takes expected linear time. As noted above the cost of iteration $i$ is linear in the size of $S_i$. Let us show that the sizes of the sets $S_i$, $i = 1, 2, \ldots$, are expected to decrease at least geometrically. The key argument is to consider the sequence $\{d(x) : x \in S_i\}$ in non-decreasing order. When selecting at random $x_i$, all points $x$ such that $d(x) \geq d(x_i)$ are deleted. Thus, on the average at least half of the points are deleted in each iteration, and hence a geometric decrease is expected. Specifically we have

**Lemma 2.2:**

$$\mathbf{E}\left(\sum_{i=1}^{i^*} \mid S_i \mid\right) \leq 2n$$

*Proof:* Let $s_i$ be the cardinality of $S_i$. We first show by induction that $\mathbf{E}(s_i) \leq \frac{n}{2^{i-1}}$ for $i \geq 1$. By the argument given above, $\mathbf{E}(s_{i+1}) \leq s_i/2$. Therefore,

$$\mathbf{E}\left(s_{i+1}\right) = \mathbf{E}\left(\mathbf{E}\left(s_{i+1}\right)\right) \leq \mathbf{E}\left(\frac{s_i}{2}\right) = \frac{1}{2}\mathbf{E}\left(s_i\right).$$

Thus, by inductive hypothesis, $\mathbf{E}\left(s_{i+1}\right) \leq \frac{1}{2}\frac{n}{2^{i-1}} = \frac{n}{2^i}$.

By linearity of expectation,

$$\mathbf{E}\left(\sum_{i=1}^{i^*} s_i\right) \leq \mathbf{E}\left(\sum_{i=1}^{n} s_i\right) = \sum_{i=1}^{n} \mathbf{E}\left(s_i\right) \leq \sum_{i=1}^{n} \frac{n}{2^{i-1}} \leq 2n.$$

□

We therefore have,

**Theorem 2.3:** *The sieve closest-pair algorithm solves the closest-pair problem in $O(n)$ expected time and $O(n)$ space.*

## 3. The $\epsilon$-Closest Bichromatic Pair Algorithm

The sieve algorithm presented consists of two parts. First we obtain an approximation to the closest bichromatic pair within a factor of 3, in a manner similar to the closest-pair algorithm. Then we show how to improve the approximation factor to $(1+\epsilon)$, for any fixed $\epsilon$.

**Obtaining a rough approximation:** For a point $x$, let $d(x)$ be the distance from $x$ to the closest point with a different color in the current set. Observe that after picking a point $x_i$ at random and computing the distance $d(x_i)$, we can filter the current set to throw out all points $x$ with $d(x) \geq d(x_i)$; no point $x$ with $d(x) < d(x_i)/3$ is thrown though. This will discard at least half the points on the average, as can be seen by considering the vector $d(x)$ ($x$ in the current set) in a sorted order. The implementation is similar to that of the closest-pair algorithm. For each neighborhood we only need to know whether or not it contains points from at least two different colors. This suffices for answering queries of the type "does $x$ have a neighbor of different color in its neighborhood".

**Refining the approximation:** Let $d(x_{i^*})$ be the approximation computed above; i.e., $d(x_{i^*})/3 \leq \delta \leq d(x_{i^*})$ where $\delta = dist(p,q)$, $(p,q)$ being a closest bichromatic pair. Consider a mesh of size $d(x_{i^*})$. It is guaranteed that $p$ is in the neighborhood of $q$. However, unlike the case in the closest-pair algorithm, the number of points of the same color in each neighborhood is unbounded; thus, computing the distance from a point to all points of a different color in its neighborhood is too costly. Instead, we proceed as follows.

Consider a refined mesh of size $b = \epsilon\, d(x_{i^*})/9$. From each subset of points of the same color lying in the same cell of the refined mesh we take an arbitrary point as a *representative*. The distance of a representative from any point in its cell is at most $\sqrt{2}b < \epsilon\, d(x_{i^*})/6$. Therefore, if $p'$ and $q'$ are the representatives of $p$ and $q$ (respectively), we have $dist(p',q') - dist(p,q) \leq \epsilon\, d(x_{i^*})/3 \leq \epsilon\, dist(p,q)$ and therefore $dist(p',q') \leq (1+\epsilon)dist(p,q)$. It remains to find a closest bichromatic pair among the representatives.

The number of representatives in each neighborhood (in the mesh of size $d(x_{i^*})$) is constant (specifically, it is $O(1/\epsilon^2)$). (The number of distinct colored points that can be packed into the cell is constant, since the mesh was an approximation to a closest bichromatic pair.) Therefore, a closest bichromatic pair of representatives can be found in $O(n)$ time by computing the distance from each representative of one color to all representatives of different colors in its neighborhood.

We have

**Theorem 3.1:** *The sieve algorithm presented above solves the $\epsilon$-closest bichromatic pair problem in $O(n)$ expected time and $O(n)$ space, for any fixed $\epsilon$.*

## 4. Conclusions

In this paper we presented a linear time randomized algorithm for the closest-pair problem, based on a new *sieving* technique. It would be interesting to see if the sieving technique would be of use in designing a deterministic algorithm that is faster than Fortune and Hopcroft's [FH79]. The algorithm was extended to obtain a randomized linear time approximation algorithm for the closest bichromatic pair problem. Subsequent to this work, Golin, Raman, Schwarz, and Smid [GRSS93] described a randomized data structure for the *dynamic* closest pair problem. For a set of $n$ points in $k$-dimensional space, for any fixed $k$, their data structure supports a closest-pair computation in constant time; it requires $O(n)$ space, and supports each insertion or deletion in expected $O(\log n)$ time. The previously best known algorithm uses $O(n \log^k n)$ space and runs in $O(n \log^k n \log\log n)$ amortized time per update; the previously best known linear-space data structure requires $O(\sqrt{n} \log n)$ time per update. Their data structure is essentially based on the algorithm presented in this paper. The *sieving procedure* presented here is, to quote the authors, "at the heart of the dynamic algorithm."

**Acknowledgments:** We are grateful to Estie Arkin, Omer Berkman, Dave Mount, Prabhakar Raghavan, Micha Sharir, and Uzi Vishkin for useful comments.

## References

[AESW90] P. K. Agarwal, H. Edelsbrunner, O. Schwarzkopf and E. Welzl, "Euclidean minimum spanning trees and bichromatic closest pairs", *6th Symp. on Computational Geometry*, pp. 203–210, (1990).

[Ben80] J. L. Bentley, "Multidimensional divide-and-conquer", *Comm. ACM 23*, pp. 214–229, (1980).

[Ben83] M. Ben-Or, "Lower bounds for algebraic computation trees", *15th Symp. on the Theory of Computing*, pp. 80–86, (1983).

[BS76] J. L. Bentley and I. Shamos, "Divide and conquer in multidimensional space", *8th Symp. on the Theory of Computing*, pp. 220–230, (1976).

[Cla83] K. L. Clarkson, "Fast algorithms for the all nearest neighbors problem", *24th Symp. on Foundations of Computer Science*, pp. 226–232, (1983).

[CLR90] T. Cormen, C. Leiserson and R. Rivest, "Introduction to Algorithms", McGraw Hill and The MIT Press, (1990).

[CW79] J. L. Carter and M. N. Wegman, "Universal classes of hash functions", *J. Computer and System Sciences*, 18, pp. 143–154, (1979).

[FH79] S. Fortune and J. E. Hopcroft, "A note on Rabin's nearest-neighbor algorithm", *Information Processing Letters*, 8(1), pp. 20–23, (1979).

[FKS84] M. L. Fredman, J. Komlós, and E. Szemerédi, "Storing a sparse table with $O(1)$ worst case access time", *J. of the Association for Computing Machinery*, 31, pp. 538–544, (1984).

[GRSS93] M. Golin, R. Raman, C. Schwarz and M. Smid, "Randomized data structures for the dynamic closest-pair problem", *4th Annual Symp. on Discrete Algorithms*, pp. 301–310, (1993).

[HNS88] K. Hinrichs, J. Nievergelt and P. Schorn, "Plane-sweep solves the closest pair problem elegantly", *Information Processing Letters*, 26, pp. 255-261, (1988).

[Knu73] D. E. Knuth, "The Art of Computer Programming, Volume 3: Sorting and Searching", Addison-Wesley, (1973).

[Mat93] Y. Matias, "Semi-dynamic closest-pair algorithms", *5th Canadian Conf. on Computational Geometry*, (1993).

[Man89] U. Manber, "Introduction to Algorithms: A Creative Approach", Addison Wesley, (1989).

[PS86] F. Preparata and M. Shamos, "Computational Geometry", Springer Verlag, (1986).

[Rab76] M. Rabin, "Probabilistic Algorithms", in *Algorithms and Complexity, Recent Results and New Directions*, Academic Press, pp. 21–39, (1976).

[SH75] M. I. Shamos and D. Hoey, "Closest-point problems", *16th Annual Symposium on Foundations of Computer Science*, pp. 151–162, (1975).

[SSS92] C. Schwarz, M. Smid and J. Snoeyink, "An optimal algorithm for the on-line closest pair problem", *8th Symp. on Computational Geometry*, pp. 330–336, (1992).

[Vai89] P. M. Vaidya, "An $O(n \log n)$ algorithm for the all-nearest-neighbors problem", *Discrete & Comput. Geometry*, 4, pp. 101–115, (1989).

[Yao91] A. C. Yao, "Lower bounds for algebraic computation trees with integer inputs", *SIAM J. Comput.*, 20:4, pp. 655–668, (1991).