Sorting Pairs of Points Based on Their Distances

Mohammad Farshi, Abolfazl Poureidi, Zorieh Soltani
Combinatorial and Geometric Algorithms Lab., Department of Computer Science
Yazd University, Yazd, P. O. Box 89195-741, Iran

***Abstract*—Sorting data is one of the main problems in computer science which studied vastly and used in several places. In several geometric problems, like problems on point sets or lines in the plane or Euclidean space with higher dimensions, the problem of sorting pairs of points based on the distance between them is used. Using general sorting algorithms, sorting $\binom{n}{2}$ distances between n points can be done in $\mathcal{O}(n^2 \log n)$ time. Ofcourse, sorting $\Theta(n^2)$ independent numbers does not have a faster solution, but since we have dependency between numbers in this case, finding a faster algorithm or showing that the problem in this case has $\Omega(n^2 \log n)$ time complexity is interesting. In this paper, we try to answer this question.**

***Keywords*—Sorting problem; Sorting distances**

I. Introduction

Sorting problem is one of the fundamental problems in computer science which studied vastly in complexity theory and several efficient algorithms proposed for it. This problem has several applications in other problems and any result on this problem, directly affect the problems using it. Basic problems, like searching, finding the closest pair, constructing the minimum spanning tree, computing the convex hull, selecting kth smallest number which has vast applications fields like implementing search engines, making road maps, air-traffic control, computer graphics and robotics, all use sorting algorithms.

As we know, all (compare-based) sorting algorithms has $\Omega(n \log n)$ time complexity and therefore, other algorithms that use sorting, has the same lower bound on their complexity. The greedy algorithm for constructing geometric spanners is one of these algorithms [1], [2]. One point in some of these problems, like the greedy algorithm for computing geometric spanners, is that they need to sort some numbers that comes from distances between all pairs of input points. So the lower bound of sorting problem does not apply for sorting distances. So in this paper, we introduce a variant of sorting problem and try to solve it. However, we can not find the answer to the questions arise about this new problem, but we has a hope that one can use this technique to answer the questions.

Problem: Given a set $S = \{p_1, p_2, \ldots, p_n\} \subset \mathbb{R}^d$, sort all pairs (p_i, p_j) based on the Euclidean distance between them.

Using the general sorting algorithms, one can sort these $\binom{n}{2} = \Theta(n^2)$ pairs of points in $\mathcal{O}(n^2 \log n)$ time and the problem has $\Omega(n^2 \log n)$ lower bound, if the numbers are independent. But in this case, the numbers are not independent, they are distances between pairs of n input points. Now the question is that, is there an $o(n^2 \log n)$ algorithm to sort the distances or this problem has $\Omega(n^2 \log n)$ time complexity.

In this paper, we study this problem in its simplest case, when the point are from one-dimensional Euclidean space, i.e. real line. We tried to find an $o(n^2 \log n)$ time algorithm for this case, but we could not succeed. So, we try to show that this problem has $\Omega(n^2 \log n)$ time complexity.

II. Decision tree for sorting distances

To show a lower bound for the time complexity of the problem, we use a usual way as used in textbooks, see [3, Chapter 8]. In this way, we consider all permutations for sorting $\binom{n}{2}$ distances between n points on $\mathbb{R}$ and then we construct a decision tree on the permutations. The depth of the decision tree is the lower bound on the time complexity of the problem. The major issue in constructing such a decision problem is the following: because there is dependency between the distances between pairs of points, some of the permutations of the distances does not happen at all. This means that we have to remove them from the set of permutations. If the remaining permutation still is large enough, then we can show that the problem has $\Omega(n^2 \log n)$ time complexity, but if the size of remaining permutation is very low, there is a hope to find an algorithm with lower time complexity. So, in the following, we try to construct a decision tree such that, the leaves of the tree corresponds to the permutations that actually can occur in sorting distances.

A. Constructing the decision tree

In a decision tree for sorting n numbers, each leaf is corresponding to a permutation of the input. If the input numbers are independent, each permutation of the input should appear in the decision tree as a leaf. This is not the case for sorting distances, some of the permutations of the distances does not appear as a leaf of the decision tree. In this section, we try to remove these permutations from the list of all permutations.

To make a decision tree, we first consider the distance matrix and mention some of its properties. This matrix helps us to ignore unnecessary comparisons in sorting distances and only perform the comparisons which are necessary. Assume we have n points $p_1, \ldots p_n$ on the real line sorted from left to right (increasing order). Obviously, the distance between p_i and all the points after it appear in the sorted list of distances between pairs of points in order that we meet the points when we start at p_i and walk to the right. In other words, for each i,

$$|p_i p_{i+1}| < |p_i p_{i+2}| < \cdots < |p_i p_n|.$$

We define the distance matrix $D = (d_{i,j})_{n \times n}$ such that $d_{i,j} = |p_i p_j|$. Obviously the distance matrix D is symmetric. So the

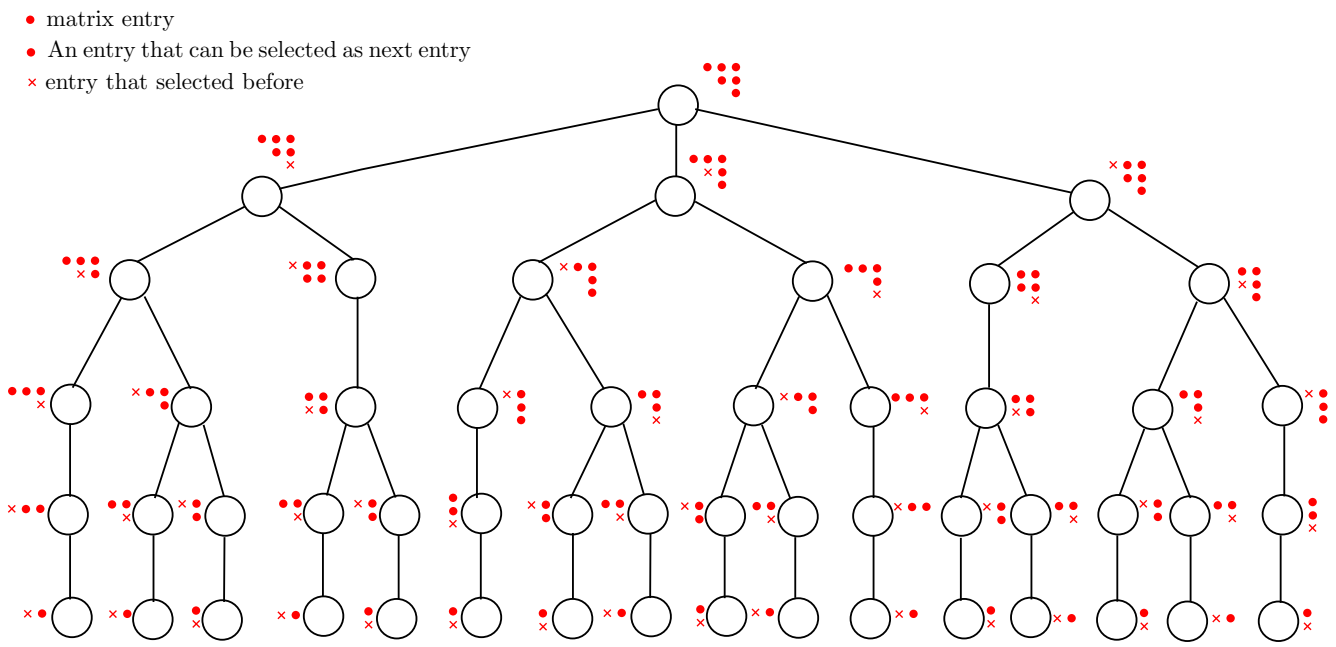

Fig. 1: The Decision tree on a set of 4 points.

matrix D is as follows (the entries below the diagonal removed because of its symmetry):

$$D = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ \vdots \\ p_{n-1} \\ p_n \end{array} \begin{array}{c} \begin{array}{cccccc} p_1 & p_2 & p_3 & p_4 & \cdots & p_n \end{array} \\ \begin{pmatrix} 0 & d_{1,2} & d_{1,3} & d_{1,4} & \cdots & d_{1,n} \\ & 0 & d_{2,3} & d_{2,4} & \cdots & d_{2,n} \\ & & 0 & d_{3,4} & \cdots & d_{3,n} \\ & & & & \vdots & \\ & & & & 0 & d_{n-1,n} \\ & & & & & 0 \end{pmatrix} \end{array}$$

Since the points are on real line, the matrix D has the following properties:

- entries of each row increases as one one moves from left to right,
- entries of each column decreases as one one moves from top to bottom.

So in sorting distances, we do not need to compare entries that are in a row or a column, because we know their position in the sorted list based on their position in the matrix. As we will see in the next section, there are some other unnecessary comparison that does not have this property. Note that one can merge the rows of the matrix to get the sorted list of distances. This will gives us an $\mathcal{O}(n^2 \log n)$ algorithm because we have n lists and a total of $\Theta(n^2)$ numbers. In the rest of paper, we denote entities of the distance matrix by dots, because its value is clear from the position of the point in the matrix (see Fig. 2).

Because of the properties of the matrix D, the smallest (non-zero) entry of each row is the first element of the row after the entry on the diagonal of the matrix. So to find the smallest distance between pairs of points, it is sufficient to find the smallest element between these entries. We remove

$$D = \begin{pmatrix} 0 & d_{1,2} & d_{1,3} & d_{1,4} & d_{1,5} \\ & 0 & d_{2,3} & d_{2,4} & d_{2,5} \\ & & 0 & d_{3,4} & d_{3,5} \\ & & & 0 & d_{4,5} \\ & & & & 0 \end{pmatrix}$$

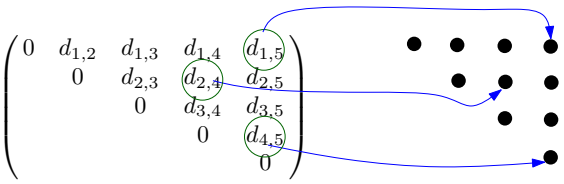

Fig. 2: Representing the entries of matrix D.

the smallest element from the list of candidates and report it as the first element of the sorted list and add the entry right after it to the candidate list. An important point here is that an entry of the matrix can be the next element of the sorted list, if there was no unselected entry on the left and below the entry.

As we mentioned before, leaves of a decision tree correspond to permutations of the (pairs of points) input such that this permutation of input can happen as a sorted list of the input. Obviously, all permutations of distances between the input points can not happen in the procedure of sorting. For example, any permutation that has $d_{1,3}$ before $d_{1,2}$ can not happen in sorting of distances, because $d_{1,3} > d_{1,2}$.

To bound the number permutation that may happen in sorting the distances, we construct a tree as follows. In the root of the tree, we do a comparison on the first entry of each row, which makes a list of $\mathcal{O}(n)$ elements. We choose the smallest element in the root and based on the element chooses in the root, we goes to the second level of the tree. Since any of the $n-1$ element of the first list can be the smallest one, we add $n-1$ children to the root. The second smallest element recognizes in the second level of the tree. The rest of the tree is constructed in a similar way. Fig. 1 shows the tree for a set of 4 points on the real line.

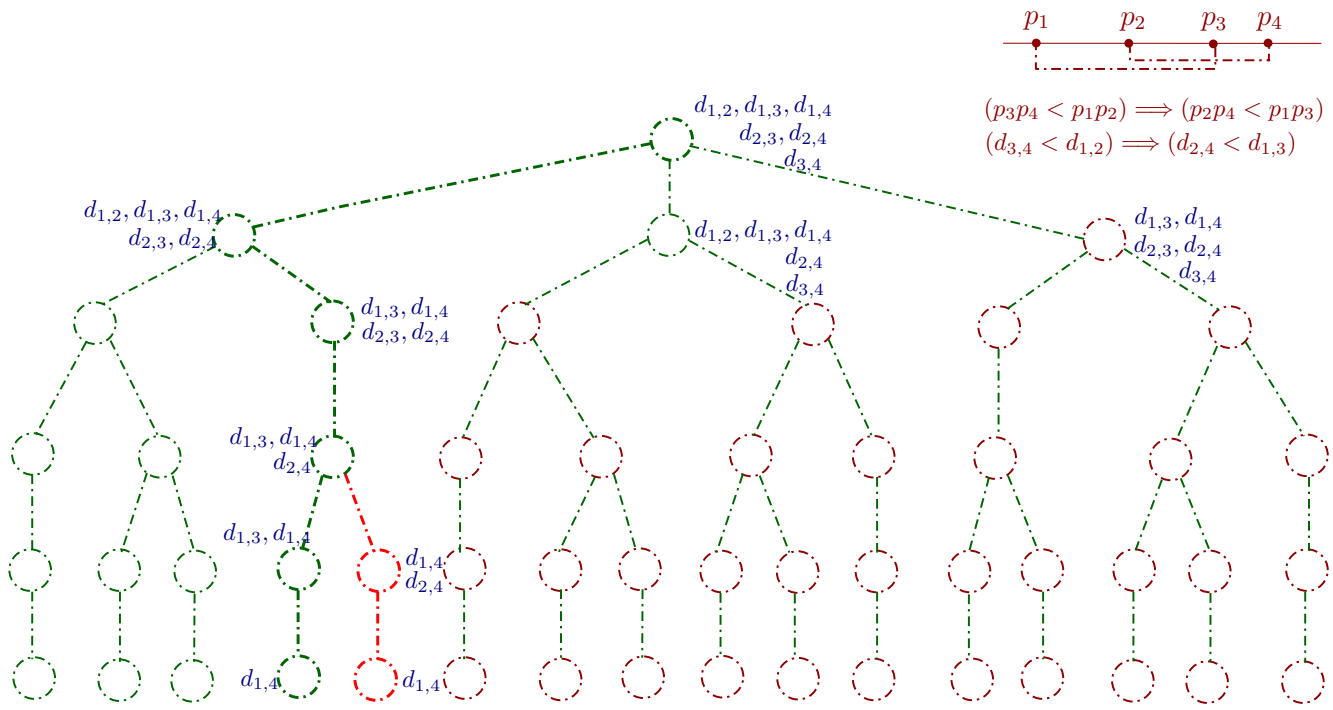

Fig. 3: One unnecessary branch of the decision tree.

It is easy to see that we have all the possible permutations that can come-up at the end of sorting process in the decision tree. At the first view, one may claim that all the permutations that we have in this tree can happens in sorting distances, but this is not correct. As one can see in Fig. 3, for sorting distances between 4 points on the real line, after the comparison of $d_{3,4} < d_{1,2}$, we can conclude $d_{2,4} < d_{1,3}$ and so we do not need to compare them. So the red branch of the tree in Fig. 3 is not necessary. If we carefully check these red branches and then remove them from the tree, we can have only the necessary permutations.

In short, one should remove all unnecessary permutations of distances from the tree and find the number of remaining permutations. If this number is big enough, it gives the desired lower bound. Otherwise, one can have a hope that an $o(n^2 \log n)$ algorithm exists for sorting all distances between n points on the real line.

B. Computing the lower bound of the number of permutations

Based on the results in the previous section, all of the leaves of the tree are not necessary, but in this section we work on bounding the number of leaves in the tree.

Computing the exact number of permutations (or number of leaves of the tree) is difficult, because the degree of inner vertices of the tree are different (maximum degree is for the root which is $n-1$ and minimum degree is 1), so we find a lower bound on the number of leaves of the tree. If this lower bound is of order of n^{n^2}, then we can conclude that the lower bound of the problem of sorting pairs of points is $\Omega(n^2 \log n)$. To bound the number of leaves from below, we find the minimum degree of vertices that lies in each level of the tree and then compute the lower bound by multiplying them.

Lemma II.1. *The difference between the degree of each node of the tree (except the root) and the degree of its parent is at most one.*

Proof: Consider an arbitrary node u of the tree and its corresponding matrix. The degree of the node u is equal to the number of elements in the set of candidate entries of the matrix that can be removed in the next step. In this step, we remove an entry from the candidate set. We have three cases:

Case 1: after removing the current entry, only one new element added to the candidate set (see Fig. 4A). In this case the degree of the node and its children is the same.

Case 2: after removing the current entry, two new elements added to the candidate set (see Fig. 4B). In this case the degree of children is one more than the degree of their parent.

Case 3: after removing the current entry, no new element added to the candidate set (see Fig. 4C). In this case the degree of children is one less than the degree of their parent.

So, we are done. ■

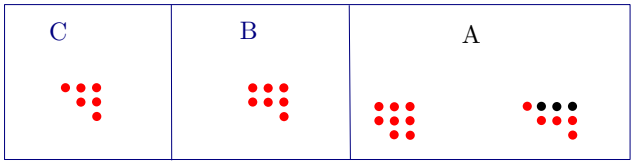

Fig. 4: Three case of the proof of Lemma II.1.

Now, we use this lemma to bound the number of leaves in the decision tree. So, in each level of the tree, we choose the node with minimum degree. On way that comes to mind is to start from the root and the by going down in the tree, we decrease the degree of the node in each step by one. This is true, but it can not go further than $(n-1)/2$th level, because if we go one step further from this level, the degree of the nodes will be zero which is unacceptable.

Considering the proof of Lemma II.1, we should choose the entries such that case 3 in the proof of the lemma happens, i.e., if we choose an entry, the next entry is chosen from a row which is at least two rows above or below the row of current entry's row (if we choose the next entry from the row just above or just below the current row, then the number of selection does not change in the next step). So, the best strategy is to choose elements from the every other row (see Fig. 5).

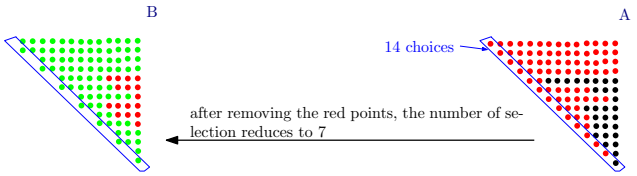

Fig. 5: The case with minimum selection for the next step.

As one can see in Fig. 5A, by removing the red entries in $(n-1)/2$ steps, the number of selection decreases to $(n-1)/2$ (the green entries in Fig. 5B). Therefore, in step $(n-1)/2+1$, if we choose any of the green entries, by Lemma II.1, the number of candidates in the next step increase by 1, i.e. $(n-1)/2+1$ selection.

As we mentioned before, we are looking for permutations (paths in the tree) such that we have the least selection for the next step. To this end, if one remove the entries showed in Fig. 7, in the order mentioned in the figure, we have $((n-1)/2)+1$, $(n-1)/2$, $(n-1)/2$ and $((n-1)/2)-1$ selection, respectively. After removing these four entries, the number of selection decreases by one and this decrease is irrecoverable. In a similar manner, we can remove the next 4 entries in similar situation which decreases the number of selections to $((n-1)/2)-2$. We continue removing the elements in this manner (see Fig. 6).

So, by summing up all of the degrees, the number of generated permutations are as follows:

$$
\begin{aligned}
&\overbrace{\frac{n}{2}\times\cdots\times\frac{n}{2}}^{\frac{n}{2}\text{ times}}\times\overbrace{\frac{n}{4}\times\cdots\times\frac{n}{4}}^{n\text{ times}}\times\overbrace{\frac{n}{8}\times\cdots\times\frac{n}{8}}^{2n\text{ times}}\times\cdots\times\\
&\cdots\overbrace{\frac{n}{2^i}\times\cdots\times\frac{n}{2^i}}^{n2^{i-2}\text{ times}}\times\cdots\times\overbrace{\frac{n}{2^{\log n}}\times\cdots\times\frac{n}{2^{\log n}}}^{n2^{\log n-2}\text{ times}}\\
=\;&(\frac{n}{2})^{\frac{n}{2}}\times\overbrace{(\frac{1}{2})^n\times(\frac{n}{2})^n}\times\overbrace{(\frac{1}{2})^{2n}\times(\frac{n}{2})^{2n}}\times\cdots\times\\
&\overbrace{(\frac{1}{2})^{n\times2^{i-2}\times(i-1)}\times(\frac{n}{2})^{n\times2^{i-2}}}\times\cdots\\
&\times\overbrace{(\frac{1}{2})^{n\times2^{\log n-2}\times(\log n-1)}\times(\frac{n}{2})^{n\times2^{\log n-2}}}\\
=\;&(\frac{n}{2})^{\sum_{i=1}^{\log n}n\times2^{i-2}}\times(\frac{1}{2})^{\sum_{i=1}^{\log n}n\times2^{i-2}\times(i-1)}\\
=\;&(\frac{n}{2})^{\frac{n}{2}\times(n-1)}\times(\frac{1}{2})^{\frac{n}{4}\times(4(1-n)+2n\times\log n)}\\
=\;&(\frac{n}{2})^{\frac{n}{2}\times(n-1)}\times(\frac{1}{2})^{n\times(1-n)}\times(\frac{1}{2})^{\frac{n^2}{2}\times\log n}\\
=\;&n^{\frac{n}{2}\times(n-1)}\times(\frac{1}{2})^{\frac{n}{2}\times(n-1)}\times(\frac{1}{2})^{n\times(1-n)}\times(\frac{1}{2})^{\log n^{\frac{n^2}{2}}}\\
=\;&n^{\frac{n}{2}\times(n-1)}\times(\frac{1}{2})^{\frac{n}{2}\times(1-n)}\times\frac{1}{n^{\frac{n^2}{2}}}\\
=\;&n^{-\frac{n}{2}}\times(\frac{1}{2})^{\frac{n}{2}\times(1-n)}=\frac{2^{\frac{n}{2}\times(n-1)}}{n^{\frac{n}{2}}}=(\frac{2^{n-1}}{n})^{\frac{n}{2}}\\
=\;&(\frac{2^{n-1}}{2^{\log n}})^{\frac{n}{2}}=2^{\frac{n}{2}\times(n-\log n-1)}.
\end{aligned}
$$

The bound is much less that the one that gives us the desired lower bound. However, we believe it is possible to find a suitable lower bound using a more sophisticated analysis of the tree.

III. Conclusion and Future Works

In this paper, we studied the time complexity of sorting distances between pairs of n input points in the real line. We could not come-up with a new result, but it seems that more sophisticated analysis of the structure that proposed in this paper will show the complexity of the problem. Our conjecture is that this problem has $\Omega(n^2\log n)$ lower bound on time complexity.

Another way of attacking the problem is to reduce another problem with known time complexity $\Omega(n^2\log n)$ to this problem. There are several algorithm in the class of quadratic time complexity (see [4], [5]). However, we could not find a problem with $\Omega(n^2\log n)$ time complexity. It is also interesting if one study the problem in higher dimensions.

References

[1] G. Narasimhan and M. Smid, *Geometric spanner networks*. Cambridge University Press, 2007.

[2] P. Bose, P. Carmi, M. Farshi, A. Maheshwari, and M. Smid, "Computing the greedy spanner in near-quadratic time," *Algorithmica*, vol. 58, no. 3, pp. 711–729, 2010. [Online]. Available: http://dx.doi.org/10.1007/s00453-009-9293-4

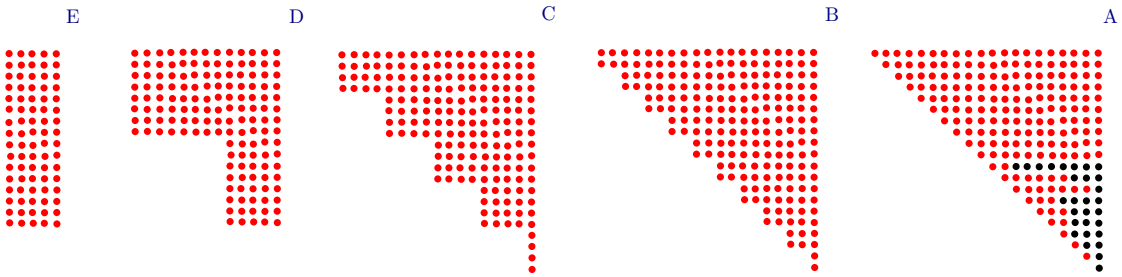

Fig. 6: The case with minimum selection for the next step.

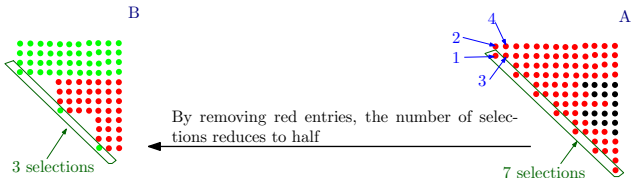

Fig. 7: The case with minimum selection for the next step.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. The MIT Press, 2009.

[4] A. Gajentaan and M. H. Overmars, "On a class of problems in computational geometry," *Computational Geometry*, vol. 45, no. 4, pp. 140 – 152, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925772111000927

[5] J. King, "A survey of 3sum-hard problems," 2004.