On Computing the Centroid of the Vertices of an Arrangement and Related Problems

Deepak Ajwani, Saurabh Ray, Raimund Seidel, and Hans Raj Tiwary *

Max-Planck-Institut für Informatik, Saarbrücken, Germany
Universität des Saarlandes, Saarbrücken, Germany

Abstract. We consider the problem of computing the centroid of all the vertices in a non-degenerate arrangement of n lines. The trivial approach requires the enumeration of all $\binom{n}{2}$ vertices. We present an $\mathcal{O}(n \log^2 n)$ algorithm for computing this centroid. For arrangements of n segments we give an $\mathcal{O}(n^{\frac{4}{3}+\epsilon})$ algorithm for computing the centroid of its vertices. For the special case that all the segments of the arrangement are chords of a simply connected planar region we achieve an $\mathcal{O}(n \log^5 n)$ time bound. Our bounds also generalize to certain natural weighted versions of those problems.

1 Introduction

An arrangement of n lines in the plane has up to $\binom{n}{2}$ vertices. However, these vertices are implicitly specified by only $2n$ real numbers. Thus it is not necessarily surprising that some functions of this vertex set can be computed in subquadratic time: E.g. the vertex with k-th smallest x-coordinate can be computed in $\mathcal{O}(n \log n)$ time [4]. It is an outstanding open problem in computational geometry whether in subquadratic time the true number of vertices can be computed (in other words, whether in subquadratic time degeneracy can be determined).

In this paper we study the problem of computing the centroid of the vertices in an arrangement of lines (and also of line segments). In contrast to the problems mentioned above the centroid function is not combinatorial in the sense that it does not produce an integer value but it produces real values.

We first show that the centroid of intersection points n lines in the plane can be computed in $\mathcal{O}(n \log^2 n)$ time. Using this result and employing a segment query data structure, we show that the centroid of the intersection points of n line segments in the plane can be computed in $\mathcal{O}(n^{4/3+\delta})$ time ($\delta > 0$ arbitrarily small). This should be compared with the complexity of the best known algorithm for counting the number of intersections in the plane by Chazelle [3], which is $\Theta(n^{4/3}(\log n)^{1/3})$. In case the segments have a restricted structure in that they all are chords of a simply connected region, we can do better: we show a bound of $\mathcal{O}(n \log^5 n)$ time. We finally show that all the mentioned bounds continue to hold for a natural weighted generalization of the centroid problem:

* The last author was supported by Graduiertenkolleg der FR Informatik, Universität des Saarlandes while working on this problem.

endow each line (or segment) with a real weight and define the weight of an intersection point to be the sum of the weights of the involved lines.

The main computational ingredient in our approach besides the usual computational geometry machinery is the Fast Fourier Transform.

Our approach does in no way solve the above-mentioned degeneracy problem. However, in the conclusion we offer a brief discussion why algebraic methods as used in this paper may be a viable approach towards a subquadratic solution the degeneracy problem For the sake of presentation we assume non-degeneracy. Degeneracy in the form of non-intersecting lines can easily be taken care of explicitly. Degeneracy in the form of of concurrent lines is ignored in the sense that if an arrangement vertex v is incident to k lines then it is counted $\binom{k}{2}$ times, once for each pair of lines intersecting in v.

The problem of computing the centroid of the vertices of an arrangement is admittedly somewhat academic. For readers with a strong need for applications here is a conceivable scenario where our results would be relevant. Consider the deployment of wireless devices on road-crossings in a city for the purpose of traffic monitoring (finding traffic rule violations or updating the people about overcrowded crossings or traffic jams). These devices need to continously transmit the data to a central base station. An important cost criterion here is the power consumed by these devices. The power needed by a device is proportional to the square (assuming free space) of the distance to which it needs to transmit the data. Thus the location of the central base station should be such that it minimizes the sum of the squares of the distances to the road crossings. This location is realized by the centroid of the crossings. Thus our results apply if all the roads in the city are straight and all intersections are crossings of exactly two roads.

If you assume that the power consumption of the wireless device at an intersection is proportional to the number of cars going by and the average number w_i of cars going along road i per unit of time is independent of the position along the road, then the weighted versions of our centroid problems apply.

Our results heavily rely on the following two facts from computational algebra (see Chapter 1 of [9], [5]).

Fact 1 (Fast polynomial multiplication) *The product of two univariate polynomials over the reals of maximal degree n can be computed in time $O(n \log n)$.*

Fact 2 (Fast multiple evaluation) *Let $p(x)$ be polynomial over the reals of degree at most n and let A be a set of n real number.*

The set $\{(a, p(a)|a \in A\}$ can be computed in $O(n \log^2 n)$ time.

2 Computing the centroid of the intersection points of n lines

We are given a set L of n lines $l_i : y = m_i x - c_i$ (for $1 \leq i \leq n$) in general position (no three of them intersect at the same point and no two of them are

parallel). Let (X_{ij}, Y_{ij}) represent the intersection point of lines l_i and l_j. We want to compute the centroid (X_L, Y_L) of the intersection points (X_{ij}, Y_{ij}). By the definition of centroid,

$$X_L = \binom{n}{2}^{-1} \sum_{\substack{i,j \in [1 \ldots n] \\ i<j}} X_{ij}, \qquad Y_L = \binom{n}{2}^{-1} \sum_{\substack{i,j \in [1 \ldots n] \\ i<j}} Y_{ij}$$

Consider a query line $l \ : \ y = \mu x - \gamma$. We would like to compute the sum of the x-coordinates of the intersection points of l with each of the lines in L. This is given by

$$F_L(\mu, \gamma) = \sum_{1 \leq i \leq n} \frac{c_i - \gamma}{m_i - \mu}$$

This function can be represented as:

$$F_L(\mu, \gamma) = \frac{P_L(\mu) - \gamma Q_L(\mu)}{S_L(\mu)} \qquad (*)$$

where P_L, Q_L and S_L are single variable polynomials of degree at most n.

We assume that the query line is not parallel to any of the lines in L. We do, however, allow it to be identical to one of the lines in L in which case we want to compute the sum of the x coordinates of the intersection of l with the other lines in L. If l is not identical to any of the lines in L, then $F_L(\mu, \gamma)$ as defined above is well defined. If l is identical to one of the lines (l_j) in L, then $F(\mu, \gamma)$ is of the form $\frac{0}{0}$ and thus F cannot be evaluated using (*). We therefore evaluate F at $\mu = m_j, \gamma = c_j$ applying de l'Hôpital's rule, which yeilds

$$F_L(m_j, c_j) = \frac{P'_L(m_j) - c_j Q'_L(m_j)}{S'_L(m_j)},$$

where P'_L, Q'_L, and S'_L denote the derivatives of P_L, Q_L, and L_L with respect to μ. We will associate the polynomials P_L, Q_L and S_L defined above with the set L of lines.

Lemma 1. *Given a set of n lines L, the associated polynomials P_L, Q_L and S_L can be computed in $\mathcal{O}(n \log^2 n)$ time.*

Proof. We arbitrarily color half the lines blue and the rest red. Let R and B be the set of red and blue lines respectively. We then recursively compute P_R, Q_R and S_R for the red lines and P_B, Q_B and S_B for the blue lines. Then,

$$\begin{aligned} F_L(\mu, \gamma) &= \frac{P_R(\mu) - \gamma Q_R(\mu)}{S_R(\mu)} + \frac{P_B(\mu) - \gamma Q_B(\mu)}{S_B(\mu)} \\ &= \frac{(P_R(\mu) \cdot S_B(\mu) + P_B(\mu) \cdot S_R(\mu)) - \gamma(Q_R(\mu) \cdot S_B(\mu) + Q_B(\mu) \cdot S_R(\mu))}{S_R(\mu) \cdot S_B(\mu)} \end{aligned}$$

Therefore,

$$P_L(\mu) = P_R(\mu) \cdot S_B(\mu) + P_B(\mu) \cdot S_R(\mu)$$
$$Q_L(\mu) = Q_R(\mu) \cdot S_B(\mu) + Q_B(\mu) \cdot S_R(\mu)$$
$$S_L(\mu) = S_R(\mu) \cdot S_B(\mu)$$

Since two polynomials of degree at most n can be multiplied in $\mathcal{O}(n \log n)$ time using FFT (Fact 1), P_L, Q_L and S_L can be computed in $\mathcal{O}(n \log n)$ time from P_R, Q_R, S_R, P_B, Q_B and S_B. Therefore P_L, Q_L and S_L can be computed in $\mathcal{O}(n \log^2 n)$ time (we get one log factor due to recursion).

Theorem 3. *Given a set R of n red lines and a set B of n blue lines such that no red line is parallel (or identical) to any blue line, we can compute the centroid (X_{RB}, Y_{RB}) of red-blue intersection points in $\mathcal{O}(n \log^2 n)$ time.*

Proof. We shall treat each of the red lines as a query line and compute the sum of the x-coordinates of its intersection with the blue lines. We then add the sums for all the red lines and the divide by the number of red-blue intersections (n^2) to get the x-coordinate of the centroid of the red-blue intersections. Since none of the red lines are identical to any of the blue lines, the x-coordinate X_L of the centroid of red-blue intersections is

$$X_L = n^{-2} \sum_{l_i \in R} F_B(m_i, c_i) = n^{-2} \sum_{l_i \in R} \frac{P_B(m_i) - c_i Q_B(m_i)}{S_B(m_i)}$$

Since the polynomials P_B, Q_B and S_B can be computed in $\mathcal{O}(n \log^2 n)$ time using Lemma 1 and also they can be evaluated at the n m_i's corresponding to the n red lines in $\mathcal{O}(n \log^2 n)$ time using FFT (See Fact 2), the overall time for the computation of X_L is $\mathcal{O}(n \log^2 n)$. The y-coordinate of the centroid of red-blue intersections can be computed similarly.

Corollary 1. *Given a set R of r red lines and a set B of b blue lines, the centroid (X_{RB}, Y_{RB}) of the red-blue intersections can be computed in $\mathcal{O}((r + b) \log^3 (r + b))$ time.*

Theorem 4. *Given a set L of n lines, we can compute the centroid (X_L, Y_L) of their intersections in $\mathcal{O}(n \log^2 n)$ time.*

Proof. In this case, we treat each of the lines as a query line and compute the sum of the x-coordinates of its intersections with the lines in L. We know that each of these query lines is identical to exactly one line in L (i.e. the line itself). Therefore, the x-coordinate of the centroid of the intersections of the lines is given by,

$$X_L = n^{-2} \sum_{l_i \in L} F_L(m_i, c_i) = \sum_{l_i \in L} \frac{P'_L(m_i) - c_i Q'_L(m_i)}{S'_L(m_i)}$$

Since P_L, Q_L and S_L can be computed in $\mathcal{O}(n \log^2 n)$ time, P'_L, Q'_L and S'_L can also be computed in $\mathcal{O}(n \log^2 n)$ time. Also, P'_L, Q'_L and S'_L can be evaluated at

the n m_i's corresponding to the n lines in $\mathcal{O}(n \log^2 n)$ time. Therefore, X_L can be computed in $\mathcal{O}(n \log^2 n)$ time. Similarly, Y_L can be computed in $\mathcal{O}(n \log^2 n)$ time.

If each of the lines $l_i \in L$ of n lines is endowed with a weight w_i, we define the centroid of weighted lines to be

$$(X_L, Y_L) = (\sum_{\substack{l_i, l_j \in L \\ i<j}} X_{ij}(w_i + w_j), \sum_{\substack{l_i, l_j \in L \\ i<j}} Y_{ij}(w_i + w_j))$$

where (X_{ij}, Y_{ij}) is the intersection point of l_i and l_j.

We proceed exactly as in the unweighted case by considering a query line $l : y = \mu x - \gamma$ with weight ω. Then, the weighted sum of the x-coordinates of the intersections of l with the lines in L is given by

$$G_L(\mu, \gamma, \omega) = \sum_{1 \leq i \leq n} \frac{c_i - \gamma}{m_i - \mu} \cdot (\omega + w_i)$$

This function can again be represented as:

$$G_L(\mu, \gamma, \omega) = \omega \cdot \frac{P_L(\mu) - \gamma Q_L(\mu)}{S_L(\mu)} + \frac{U_L(\mu) - \gamma V_L(\mu)}{S_L(\mu)}$$

where P_L, Q_L and S_L are as before and U_L and V_L are some other polynomials of degree at most n in μ.

We can now apply the same techniques as for the unweighted case to obtain the following:

Lemma 2. *Given a set R of n red weighted lines and a set B of n blue weighted lines and their associated polynomials the centroid of the red-blue intersections can be computed in $\mathcal{O}(n \log^2 n)$.*

Theorem 5. *Given a set L of n weighted lines the centroid (X_L, Y_L) of their intersections can be computed in $\mathcal{O}(n \log^2 n)$ time.*

Corollary 2. *Given a set R of r weighted red lines and a set B of b weighted blue lines, the centroid (X_{RB}, Y_{RB}) of the red-blue intersections can be computed in $\mathcal{O}((r+b) \log^2 (r+b))$ time.*

3 Centroid of line segment intersections

Theorem 6. *Given a set R of r red segments and a set B of b blue segments, if R can be preprocessed into a data structure of size $\mathcal{O}(s(r))$ in time $\mathcal{O}(p(r))$ so that all segments intersecting a query segment $t \in B$ can be reported as the union of $\mathcal{O}(u(r))$ "canonical" prestored subsets in $\mathcal{O}(q(r))$ time, then we can form sets $R_1, R_2, \cdots, R_k$ of red segments and sets $B_1, B_2, \cdots, B_k$ of blue segments in $\mathcal{O}(p(r) + b\ q(r))$ time such that*

1. *For $1 \leq i \leq k$, all segments in R_i intersect all segments in B_i*
2. *If a red segment ρ intersects a blue segment β, there is a unique i such that $\rho \in R_i$ and $\beta \in B_i$*
3. $\sum_{i=1}^{n} |R_i| = \mathcal{O}(s(r))$, $\sum_{i=1}^{n} |B_i| = \mathcal{O}(b\ u(r))$

Proof. We use a method used by Agarwal and Varadarajan in [2]. We construct the data structure for the red segments. The canonical prestored subsets produced by the data structure form our sets R_i. With each set R_i we associate a bucket which is initially empty. We query this data structure for each blue segment $\beta \in B$ one by one. The output of the query is given as the disjoint union of $\mathcal{O}(u(r))$ canonical subsets and we put the segment β into the buckets associated with each of those subsets. The set of segments in the bucket associated with R_i forms the set B_i. It is clear that each segment in R_i intersects each segment in B_i since we put exactly those segments in B_i which intersect all segments in R_i. Also, since the output to each query is given as a disjoint union of canonical subsets, whenever a red segment ρ and a blue segment β intersect, there is a unique i such that $\rho \in R_i$ and $\beta \in B_i$. Since the size of the data structure is $\mathcal{O}(s(r))$, $\sum_{i=1}^{n} |R_i| = \mathcal{O}(s(r))$ and since each query returns $\mathcal{O}(u(r))$ canonical subsets, each blue segment is contained in at most $\mathcal{O}(u(r))$ buckets and therefore $\sum_{i=1}^{n} |B_i| = \mathcal{O}(b\ u(r))$. The time required to construct the data structure is $\mathcal{O}(p(r))$ and the time for the b queries is $\mathcal{O}(b\ q(r))$. So, the total time complexity is $\mathcal{O}(p(r) + b\ q(r))$.

Theorem 7. *Given a set R of r (weighted) red segments and a set B of b (weighted) blue segments, if R can be preprocessed into a data structure of size $\mathcal{O}(s(r))$ in time $\mathcal{O}(p(r))$ so that all segments intersecting a query segment $t \in B$ can be reported as the union of $\mathcal{O}(u(r))$ "canonical" prestored subsets in $\mathcal{O}(q(r))$ time, then we can compute the number of the red-blue intersections and their centroid in $\mathcal{O}\left(p(r) + b\ q(r) + (s(r) + b\ u(r)) \log^2 (r + b)\right)$ time.*

Proof. We use Theorem 6 to form the sets R_i and B_i. Since all segments in R_i intersect all segments in B_i, the total number of intersections between the segments in R_i and the segments in B_i is $m_i = |R_i||B_i|$ and the centroid of those intersections (X_i, Y_i) can be computed in $\mathcal{O}(n_i \log^2 n_i)$ time (Corollaries 1 and 2) where $n_i = |R_i| + |B_i|$. So, the total number of intersections is $m = \sum_i m_i$ and the centroid of all red-blue intersections is $m^{-1} \sum_i m_i X_i$ and can be computed in

$$\mathcal{O}\left(p(r) + b\ q(r) + \sum_i n_i \log^2 n_i\right) = \mathcal{O}\left(p(r) + b\ q(r) + \log^2 (r + b)(\sum_i n_i)\right) =$$

$\mathcal{O}\left(p(r) + b\ q(r) + (s(r) + b\ u(r)) \log^2 (r + b)\right)$ time.

Theorem 8. *If a set S of n (weighted) segments in the plane can be preprocessed into a data structure of size $\mathcal{O}(s(n))$ in time $\mathcal{O}(p(n))$ so that all segments intersecting a query segment $t \in S$ can be reported as the union of $O(u(n))$ "canonical" prestored subsets in $\mathcal{O}(q(n))$ time, then we can compute the number of segment intersections and their centroid in $\mathcal{O}((p(n) + n\ q(n)) \log n + (s(n) + n\ u(n)) \log^4 n)$ time.*

Proof. We color half the segments red and the rest blue and then use Theorem 7 with $r = b = n/2$ to compute the number m_{RB} of red-blue intersections and their centroid (X_{RB}, Y_{RB}). We recursively compute the number m_R of red-red intersections and their centroid (X_R, Y_R) and also the number m_B of blue-blue intersections and their centroid (X_B, Y_B). The total number of intersections is $m = m_{RB} + m_R + m_B$ and their centroid is $(m^{-1}(m_{RB}X_{RB} + m_R X_R + m_B X_B), (m^{-1}(m_{RB}Y_{RB} + m_R Y_R + m_B Y_B))$. The time required for the computation is $\mathcal{O}((p(n) + n\ q(n)) \log n + (s(n) + n\ u(n)) \log^3 n)$ (we get an extra $\log n$ factor due to the recursion).

Corollary 3. *The centroid of the intersections of arbitrary (weighted) segments in the plane can be computed in $\mathcal{O}(n^{4/3+\epsilon})$ time.*

Proof. Agarwal and Sharir [1] have shown that given a collection S of segments in the plane and a parameter $n^{1+\epsilon} \leq s \leq n^{2+\epsilon}$ we can preprocess S into a data structure of size s, in time $\mathcal{O}(s^{1+\epsilon})$ so that we can report all k segments of S intersecting a query segment in time $\mathcal{O}(n^{1+\epsilon}/s^{\frac{1}{2}} + k)$. Furthermore, the output to such a query is given as the disjoint union of $\mathcal{O}(n^{1+\epsilon}/s^{\frac{1}{2}})$ "canonical" prestored subsets. We put $s = n^{4/3}$ and apply Theorem 8 and the result follows.

Since the time complexity of the best known algorithm even for computing the number of intersections among n line segments in the plane is $\Theta(n^{4/3}(\log n)^{1/3})$ [3], it is unlikely that this can be improved.

4 Intersection of lines inside a polygon

If the configuration of the given set of segments allows a better query structure for segment intersections, we can do better. This for example is the case when the segments are chords of a simple region.

A region $D \subset \Re^2$ is called simple if any line intersects it at most c(a constant) times and the intersection of any set of n lines and the boundary of D can be ordered along the boundary in $\mathcal{O}(n \log n)$ time [8]. A chord of the region is a segment joining two boundary points and lying completely in the interior of the region. Let $c_1, c_2, \cdots, c_n$ be chords of a simple region R. We represent a chord c_i with the pair (l_i, r_i) where l_i and r_i are indices of the endpoints of c_i in the sorted order and $l_i < r_i$. Now given a query chord c, we can do binary searching of its endpoints in the sorted order of endpoints to compute their ranks l_c and r_c in $\mathcal{O}(\log n)$ time. Denote by $I(c)$ the set of chords intersecting c. Observe that $I(c) = I_1(c) \cup I_2(c)$ where $I_1(c) = \{c_i | l_i < l_c\ \&\ l_c < r_i < r_c\}$ and

$I_2(c) = \{c_i | l_c < l_i < r_c \ \& \ r_i > r_c\}$. If we represent c_i by the point (l_i, r_i) in the plane, then I_1 and I_2 can be computed using orthogonal range queries of the types $[-\infty, l_c] \times [l_c, r_c]$ and $[l_c, r_c] \times [r_c, \infty]$. We can use range trees [6], which can be built in $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n \log n)$ space, to answer such queries in $\mathcal{O}(\log^2 n)$ time where the answer to each query is the union of $\mathcal{O}(\log^2 n)$ pairwise disjoint canonical subsets. We can therefore apply Theorem 8 (with $p(n) = \mathcal{O}(n \log n), s(n) = \mathcal{O}(n \log n), u(n) = \mathcal{O}(\log^2 n), q(n) = \mathcal{O}(\log^2 n)$) to show that the centroid of the intersections of the chords can be computed in $\mathcal{O}(n \log^5 n)$ time.

Corollary 4. *The centroid of the intersection points of n (weighted) lines lying inside a given simple region D can be computed in $\mathcal{O}(n \log^5 n)$ time.*

Proof. Since each line intersects D at most c times, the problem can be reduced to computing the centroid of the intersections of at most $\lfloor c/2 \rfloor n$ chords of D.

5 Computing Higher Moments

The x and y coordinates of the centroid of the intersection points are the averages of the coorresponding coordinates of the intersection points. We are now interested in the averages of the higher powers of the x and y coordinates.

As in Section 2, consider a set L of n lines and a query line $l : y = \mu x - \gamma$. We want to compute the averages of the k^{th} powers, $X_L^{(k)}$ and $Y_L^{(k)}$, of the x and y-coordinates of the intersection points of the line l with the lines in L. The sum of the the k^{th} powers of the x-coordinates of these intersection points is

$$F_L^{(k)}(\mu, \gamma) = \sum_{1 \le i \le n} \left(\frac{c_i - \gamma}{m_i - \mu} \right)^k$$

This function can be represented as:

$$F_L^{(k)}(\mu, \gamma) = \frac{\sum_{0 \le j \le k} \gamma^j P_L^{(j)}(\mu)}{S_L(\mu)}$$

where the $P_L^{(j)}$'s and S_L are polynomials of degree at most kn in μ.

We can therefore proceed as before and prove that computing the averages of the k^{th} powers of the coordinates requires at most k^2 times the time required to compute the centroid (We get one k since there are $\mathcal{O}(k)$ polynomials to deal with and another k since the polynomials are of degree kn).

6 Conclusion and Outlook

In this paper we describe subquadratic algorithms for computing certain functions on the set of all intersecting pairs of lines (or segments) in an arrangement. Which functions do admit such subquadratic computations? The most

outstanding question of this sort akss whether two distinct intersecting pairs of lines define the same intersection point, in other word, do some three lines intersect in a point. This is known as the line arrangement degeneracy problem. At this point this problem seems out of reach. However, we would like to point out the related problem of 3-SUM may be amenable to an algebraic approach as used in this paper.

The 3-SUM problem [7] asks whether for three given sets A, B, and C of n real numbers each, we have $(A + B) \cap C = \emptyset$, where $A + B$ is the Minkowski sum $\{a + b | a \in A, b \in B\}$. No subquadratic time solutions to this problem are known, except for the case where the three sets consist of integers in the range of 0 to K. In that case it suffices to compare C with the support of the product polynomial $\left(\sum_{a\in A} x^a\right) \cdot \left(\sum_{b\in B} x^b\right)$. This can be done in $O(K \log K)$ time.

Let us consider the simpler question whether we have $A \cap B = \emptyset$. There is an obvious $O(n \log n)$ solution via sorting and merging. This solution is combinatorial in that it relies on order comparisons $<, =, >$. It may be somewhat surprising that there is a also subquadratic solution relying solely on equality comparisons: Consider the polynomial $p_A(x) = \prod_{a\in A}(x - a)$. We have $A \cap B \neq \emptyset$ iff $p_A(b) = 0$ for some $b \in B$. The polynomial p_A can be computed in $O(n \log^2 n)$ time using divide-and-conquer and FFT-based polynomial multiplication, and within the same bound p_A can be evaluated for all n elements of B. Thus $A \cap B = \emptyset$ can be decided in $O(n \log^2 n)$ time without using order comparisons.

Algebraically maybe more succinct is the formulation $A \cap B = \emptyset$ iff $\gcd(p_A(x), p_B(x)) = 1$, where of course $p_B(x) = \prod_{b\in B}(x - b)$. This also leads to an $O(n \log^2 n)$ time solution since the gcd of two polynomials of degree n can be computed within this time (See Section 2.4, Chapter 2 of [9]).

The 3-SUM problem allows the following algebraic formulation:

$$(A + B) \cap C = \emptyset \text{ iff} \gcd(\text{ resultant}(p_A(x), p_B(z - x), x) \ , \ p_C(z)) = 1 \, .$$

Here $p_C(z) = \prod_{c\in C}(z - c)$. Note that resultant$(p_A(x), p_B(z - x), x)$ is nothing but $p_{A+B}(z)$, i.e. the polynomial whose roots are exactly the elements of $A + B$. Current techniques of computational algebra do not seem to allow the evaluation of gcd(resultant$(p_A(x), p_B(z - x), x)$, $p_C(z))$ in subquadratic time. However, there may be a better chance of success along such an algebraic approach than along a traditional combinatorial approach.

References

1. Pankaj K. Agarwal and Micha Sharir. Applications of a new space-partitioning technique. *Discrete Comput. Geom.*, 9(1):11–38, 1993.
2. P.K. Agarwal and K.R. Varadarajan. Efficient algorithms for approximating polygonal chains. *Discrete Computational Geometry*, 23:273–291, 2000.
3. B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete Computational Geometry*, 9(2):145–158, 1993.
4. R. Cole, J. Salowe, W. Steiger, and E. Szemerédi. Optimal slope selection. *SIAM J. Computing*, 18:792–810, 1989.

5. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms: Chapter 32*. MIT Press, 2nd edition, 2001.
6. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications: Chapter 5*. Springer, 1997.
7. Anka Gajentaan and Mark H. Overmars. On a class of $\emptyset(n^2)$ problems in computational geometry. *Comput. Geom. Theory Appl.*, 5(3):165–185, 1995.
8. S. Langerman and W. Steiger. Ham-sandwich cuts and other tasks in arrangements. 2001. Technical report.
9. Chee Keng Yap. *Fundamental problems of algorithmic algebra*. Oxford University Press, Inc., New York, NY, USA, 2000.