On k-hulls and Related Problems†

Richard Cole *Micha Sharir*‡ *Chee K. Yap*

Courant Institute of Mathematical Sciences,
New York University,
251, Mercer Street,
New York, NY 10012.

Abstract

For any set X of points (in any dimension) and any $k = 1, 2, \ldots$, we introduce the concept of the k-hull of X. This unifies the well-known notion of 'convex hulls' with the notion of 'centers' recently introduced by F.F. Yao. The concept is intimately related to some other concepts (k-belts, k-sets) studied by Edelsbrunner, Welzl, Lovász, Erdös and others.

Several computational problems related to k-hulls are studied here. Some of our algorithms are of interest in themselves because of the techniques employed; in particular, the 'parametric' searching technique of Megiddo is used in a non-trivial way. We will also extend Megiddo's technique to Las Vegas algorithms. Our results have applications to a variety of problems in computational geometry: efficient computation of the 'cut' guaranteed by the classical 'Ham Sandwich theorem', faster preprocessing time for polygon retrieval, and theoretical improvements to a problem of intersecting lines and points posed by Hopcroft.

1. Introduction

In this paper we study the computational properties of three related concepts: k-hulls, α-partitioners and the 'Ham Sandwich cut'. The naturalness of these notions is reinforced by the fact that they are intimately related to other concepts studied elsewhere. Furthermore, efficient algorithms related to these objects have applications to problems arising in different ways. In this introduction, we define these concepts.

Let X be a set of n planar points. A point p inside the convex hull of X can be characterized by the property that any line through p has at least a point of X in each of the closed half-planes determined by the line. Generalizing this, we define the *$k-hull$* ($k > 0$) of X as the set of points p such that any line through p has at least k points of X on each closed half-plane. Clearly the k-hull contains the $k+1$-hull, and if $k > n/2$ then the k-hull is empty. An easy consequence of Helly's theorem [YB] shows that the k-hull is always non-empty for $k = \lceil n/3 \rceil - 1$; the k-hull will be known as the *center* of X in this (extreme) case. Henceforth we assume $1 \le k \le \lceil n/3 \rceil - 1$ (unless otherwise noted). It is clear that these definitions may be modified for dimensions above two.

The concept of a k-hull is apparently new but it is intimately related to other concepts which have been studied. It turns out to be (essentially) the dual of the notion of 'k-belts' recently introduced by H. Edelsbrunner and E. Welzl [EW2]. The notion of 'center' is first introduced by F.F. Yao [Y]. The concept of k-sets, previously studied by Lovász and others [L,ELSS,EW1] is also (clearly) connected to k-hulls: a k-set of X is defined to be a subset of size k of the form $X \cap H$ for some half-space H. The maximum number of k-sets for a set of n points in the plane will be denoted by $N_k(n)$. These numbers are important in bounding the running time of our algorithms. The best known bounds are $N_k(n) = O(nk^{1/2})$ and $= \omega(n \log k)$ [ELSS,EW1]. A possible application of k-hulls is as a new tool in statistics: they give an alternative measure of the 'interiorness' of an arbitrary point with respect to a fixed set X. Previously, the 'k^{th} iterated convex hull' (see [OVL]) is used for this purpose; a possible disadvantage of the iterated hulls (in contrast to k-hulls) is that there may be no k^{th} iterated hull for $k > 1$.

To motivate the next concept, we recall there are 'divide-and-conquer' algorithms on a set X of points in which the 'divide' step calls for partitioning X into four (roughly) equal parts using a pair of lines. More generally, let $0 < \alpha \le 1/4$. A pair of lines

‡School of Mathematical Sciences, Tel Aviv University, Tel Aviv, Israel; partially supported by a grant from the U.S.-Israeli Binational Science Foundation.
†This work is done under the auspices of the NYU/CIMS Laboratory for Robotics and Experimental Computer Science, which is supported by grants from Digital Equipment Corporation, The Sloan Foundation, and ONR grant No. N00014-82-K-0381.

L_i ($i = 1, 2$) is called an α-*partitioner* of X if the subsets of X in each of the four quadrants (points on L_i are not counted in any of the subsets) has size at most αn. Willard [W] shows that 1/4-partitioners always exist. The previous best algorithm for computing the 1/4-partitioner takes $O(N_{n/2}(n)\log^2 n) = O(n^{3/2}\log^2 n)$ time [EW3], an improvement on Willard's solution. Yao [Y] first shows that one can find α-partitioners from centers (she actually did this in 3-dimensions). Willard and Yao needed partitioners in constructing the data-structure for their half-space retrieval algorithms (in 2 and 3 dimensions, respectively).

The connection between k-hulls and α-partitioners can be seen in the following two facts: clearly the intersection of a pair of lines which form an α-partitioner is in the αn-hull. Conversely, it is not hard to show that a point in the αn-hull is the intersection of some pair of lines which form an $\alpha/2$-partitioner.

The third and final concept studied here comes from the following classical result. The 'Ham Sandwich Theorem' (in a form relevant to us) asserts that given d disjoint sets of points in d-dimensional space, there is a hyperplane (the 'ham sandwich cut') which simultaneously divides each point set evenly. It is intuitively clear that this cut is related to partitioners. Only recently, [DE] shows that this theorem implies the existence of a 1/8-partitioner in 3-dimensions (the corresponding result in 2-dimensions is easy).

There are two techniques underlying all our algorithms. The first is a type of 'parametric' searching which converts a suitable parallel searching algorithm into an efficient sequential algorithm. This is due to Megiddo [M] although it will be seen that our use of the technique is non-obvious. Furthermore, we will extend his technique in a new way to obtain fast probabilistic (ie. Las Vegas) algorithms. The other technique† has its roots in a paper by Lovász [L]. We will show a partial generalization of Lovász's algorithm in 3-dimensions. In the remainder of this paper, we give algorithms for computing the k-hull in two and three dimensions, for finding the 'ham sandwich cut', and for finding a point in the center. These algorithms will each illustrate one of the techniques. We conclude the paper by showing several applications of these results.

We end this introduction by summarizing our results. Most of our bounds are in the worst case sense but note that some bounds are (Las Vegas) probabilistic.

† as Professor Pollack pointed out to us.

PROBLEM	PREVIOUS	NEW RESULT	REMARK
Computing the k-hull in the plane	$O(N_k(n)\log^2 n)$ [EW2]	$O(N_k(n)\log^2 k)$	[EW2] solves the dual problem.
Finding an α-partitioner ($\alpha \leq 1/4$)	$O(N_{n/2}(n)\log^2 n)$ [EW3]	$O(n\log n(\log\log n)^2)$ (deterministically) $O(n)$ (probabilistically)	Preprocessing time for polygon retrieval in [W,EW3] is now $O(n\log^2 n(\log\log n)^2)$.
Finding a point in the center	$O(n^4)$ [Y] (3-dim only)	$O(n\log^6 n)$ (2-dim) $O(n^2\log^{10} n)$ (3-dim)	Recursive application of Megiddo's technique.
Computing the k-hull in 3-dimensions		$O(n^3\log n)$	Partial generalization Lovász's algorithm.
Ham sandwich cuts (2-dim)	$O(n^3)$ (naively)	$O(n\log n(\log\log n)^2)$ (deterministically) $O(n)$ (probabilistically)	assuming unique cut
Detecting intersection of n lines with n points	$O(n^{1.5}\log n)$	$O(n^{1.437}\log n)$	posed by J. Hopcroft

2. Computing the k-hull

Our algorithm for computing the k-hull X, of a planar set of points, has an extremely simple and appealing structure, described thus. Define a (directed) line L to be a *$k-$divider* if L has $\leq k-1$ points of X strictly to its right and $\leq n-k$ points of X strictly to its left (so L must contain some point of X). Note that for any orientation $0 \leq \theta < 2\pi$, there is unique k-divider which we denote by L_θ. Assume (for simplicity) that no three points of X are colinear.

Rotating Line Algorithm. (Lovász)

Assume that the horizontal k-divider L_0 passes through a unique point p_0 of X. Let L be a 'rotating line' and p its 'pivot'. Initially, let $L := L_0$ Rotate the line L anticlockwise about p until it meets a new point q. (Note: q may be 'forward' or 'backward' with respect to p on L.) Set $p := q$ and repeat the rotation until L becomes horizontal again.

Note that this 'algorithm' leaves out details of implementing the rotation. But first let us show how it can compute the k-hull: let $\mathbf{H}$ be the collection of half-spaces where each H in $\mathbf{H}$ is the half-space to the right of L formed at the instant when the pivot changes from p to a new q with q 'forward' of p; call the k-divider at that instant a *special* k-divider, and the corresponding half-space, a *special* half-space. Define H_k to be the intersection of the half-spaces in $\mathbf{H}$. Let L_1, L_2 be two special dividers which determine consecutive sides of H_k. Let G_i ($i = 1,2$) be the closed half-plane to the right of L_i (Figure 1).

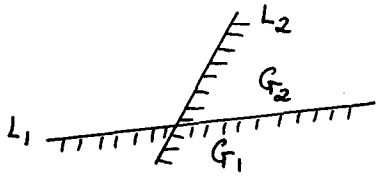

Figure 1.

Lemma 1. As L rotates from L_1 to L_2, it remains in the region $G = G_1 \cup G_2$.

Proof. It is enough to show that as L rotates from L_1 to L_2, the intersection of L and L_1 lies in the region G_2. Otherwise, if L violates this condition for the first time, it must be pivoting at a point in $G_1 - G_2$ at that moment. Then we see that it must eventually reach a special divider which intersects L_1 outside of G_2, contradicting the fact that L_1 and L_2 are consecutive sides of H_k. □

Corollary. All k-dividers lie outside H_k.

Theorem 1. The k-hull is the intersection of the half-spaces in $\mathbf{H}$.

Proof. We show that H_k is the k-hull. Clearly the k-hull is included in H_k: for if p is in the k-hull and L' is a special k-divider then p must lie either on or to the left of L'. To see the reverse inclusion, if $p \in H_k$ then any line L' through p has $\geq k$ points to its right since the k-divider with the orientation of L' must lie to the right of p, by the above corollary. □

The obvious implementation of the rotations in the above algorithm calls for a sorting of the rotational order of X about each $p \in X$, costing $O(n^2 \log n)$. This preprocessing is the most expensive part of the algorithm. Finding the k-hull takes an additional $O(N_k(n)\log n)$ time. This actually proves a stronger result:

Theorem 2. All k-hulls of X can be (simultaneously) computed in $O(n^2 \log n)$ time.

If only one k-hull is asked for, we can do better:

Theorem 3. A single k-hull can be computed in time $O(N_k(n)\log^2 k + n\log n)$ time.

Proof. Consider the rotating line L as in our original algorithm. At each instant except when the pivot changes L is pivoting about a point p and partitions $X - \{p\}$ into two sets, X_0 and X_1. Note that the next pivot point q is on the convex hull of either X_0 or X_1. Assuming that each X_i is stored in a dynamic convex hull structure of [OVL], we can easily determine q in $O(\log n)$ time. Then we update p to q, and if $q \in X_i$ ($i = 0$ or 1) we insert p and delete q from X_i, in $O(\log^2 n)$ time. Thus we take $O(N_k(n)\log^2 n)$ time to do all the rotation. Forming the intersection of the $\leq N_k(n)$ half-planes takes time $O(N_k(n)\log^2 N_k(n))$. This proves a running time of $O(N_k(n)\log^2 n)$ which is not quite theorem 3.

To improve this asymptotic bound, we first need a definition of the *$k-$iterated convex hull* (k^{th} hull, for short). If $k = 1$, this is just the usual convex hull. For $k > 1$, this is the convex hull of the point set after excluding those points in the j^{th} hull, $j = 1, 2, \ldots, k-1$. In [OVL] it is shown that all iterated convex hulls can be computed in time $O(n\log^2 n)$. Recently Chazelle has improved this to $O(n\log n)$ [C].

We now have two observations. The first is that the k-hull contains the k^{th} hull. In particular, for each k-divider there is some $r \leq k$ such that the divider intersects the j^{th} hull for $j = 1, 2, \ldots, r$ but no other i^{th} hulls. Using this fact, it is not hard to replace the sets X_i ($i = 0, 1$) used in the above algorithm by subsets $Y_i \subseteq X_i$ which have size at most $2r$. Thus the time to do a single rotation is $O(\log^2 k)$. We also have to keep track of where the current divider may cut the $r+1st$-hull. This is just a matter of determining when the divider becomes a tangent to this

hull. To do this we traverse the $r+1st$-hull, keeping track of the vertex which has a tangent parallel to the divider. Since the divider turns through an angle of 2π we go round each ith-hull at most once, so the traversal time is at most $O(n + N_k(n))$. (We only advance the vertex on the $r+1st$ hull -- if the value of r changes we then 'catch up' on the new $r+1st$-hull.)

The second observation is that we can form the intersection of the special half-planes in time $O(\log k)$ (rather than $O(\log n)$) per plane. We represent the intersection of all the special half-planes found so far in the algorithm by the sequence (in the order of their discovery)

$$l_1, l_2, \ldots, l_m \qquad (m \geq 0) \tag{2}$$

where the l_i's are the *non−occluded* special k-dividers: we say a special k-divider l is *occluded* if the special half-plane which it determines contains the intersection of all the other special half-planes found so far. Now suppose l is a newly found special k-divider. It is not hard to see that it will occlude some suffix of the sequence (2), say $l_h, l_{h+1}, \ldots, l_m$ for some $1 < h \leq m+1$ ($h = m+1$ means nothing is occluded). We claim that $m - h \leq O(k^2)$. Consider (see figure 2) any occluded special k-divider l_i, $h \leq i \leq m$.

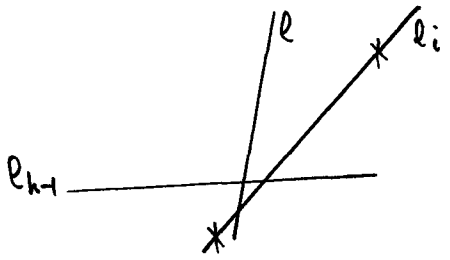

Figure 2.

It is not hard to see l_i is determined by some pair of points in X which lies to the right of l or of l_{h-1}. But there are $O(k^2)$ such pairs, proving our claim. From this we see that a binary search of the suffix of (2) of length $O(k^2)$ will determine h. This shows our theorem. *Note*. Actually, the $O(k^2)$ bound can be improved to $N_{2k}(k)$. □

This improves slightly the result of [EW2]. The crucial technique here (as well as in [EW2]) is the dynamic maintenance of the convex hull of a point set due to van Leeuwen and Overmars [OVL].

The generalization of Lovász's algorithm to higher dimensions has been mentioned as an open problem in [ELSS]. We shall give a partial generalization (in '2.5-dimension'). This can be vividly described in the following way.

A Rotating Strip Algorithm for a bicolored set of points. Let X be a set of n planar points colored either red or black. For any (oriented) line L and $p \in X$, we say p is 'good' with respect to L if p is red and strictly on the right side of L or if p is black and strictly to the left. 'Bad' points are those not on L which are not 'good' (so points on L are neither good nor bad). Call L a *$k-$divider* of X iff it has at most $k-1$ good points and at most $n-k+1$ bad ones. (In contrast to the original dividers for a monochromatic set, L need not pass through a point of X and is not unique for a given orientation.) Let $r, b \geq 0$ where $k-1 = r+b$. For a given orientation there is at most one (perhaps no) strip of k-dividers which have r good red points. We can generalize the rotating line algorithm to a 'rotating strip' ('strip' because the potential k-dividers lie within a strip) algorithm to find all the k-dividers which have r good red points, and which pass through two points of X. We say an r-divider passing through two points of X with $r = k-1$ or $r = k-2$ is a *special* k-divider. Our algorithm finds all special k-dividers (we need these to find the k-hull in 3 dimensions, as explained below).

Let $L_R(\theta, i)$ (resp. $L_B(\theta, i)$) be a i-divider (as defined for the original problem) of the red (resp. black) points with orientation θ. For $r = 0, \ldots, k-1$, $LL(\theta, r)$ denotes the (closed) strip bounded by $L_R(\theta, r+1)$ and $L_B(\theta+\pi, k-r)$. We make the following observation:

Observation. For each orientation θ, and $r = 0, \ldots, k-1$: if the interior of $LL(\theta, r)$ contains any point of $\hat{X}$, or if $L_B(\theta+\pi, k-r)$ lies strictly to the right of $L_R(\theta, r+1)$, then the set of k-dividers of $\hat{X}$ with r good red points and with orientation θ is empty. Otherwise this set consists of those lines with orientation θ lying in the strip $LL(\theta, r)$.

It follows from this observation, and from the non-degeneracy assumption on X, that for each θ and r, there is at most one special k-divider of $\hat{X}$ with orientation θ and with r good red points. We can now give a simple algorithm to find all special k-dividers with r good red points: Begin by finding $L_R(0, r+1)$ and $L_B(\pi, k-r)$. This determines three subsets of the red (resp. black) points, the subsets corresponding to points lying in the strip $LL(0, r)$ or lying to one side of the strip (red points on $L_R(0, r+1)$ and black points on $L_B(\pi, k-r)$ are not in any subset). These 6 sets are maintained in dynamic convex hull structures of [OVL]. Now we can rotate L_R and L_B (say, anticlockwise) simultaneously, updating each of the sets in the obvious manner. Each time one of these lines passes through a special k-divider of $\hat{X}$, we record its position. Each step of the rotation can be done in time $O(\log^2 n)$.

It is not hard to extend the just-described 'rotating strip' algorithm to allow the simultaneous rotation of the strips for $r = 0, \ldots, k-1$. Instead of rotating 2 parallel lines, we now rotate $2k$ parallel lines. This determines $k+1$ subsets of the red (resp. black) points which we again maintain in dynamic convex hull structures. By suitable bookkeeping of the evolving information, we can detect all special k-dividers of $\hat{X}$ as they are passed through. Each step of the rotation takes $O(\log^2 n \log k)$ time: the $\log k$ factor comes from having to take the minimum of $2k$ possible increments of the current orientation.

Let us assess the complexity of this algorithm. We count the number of 'events' where an event occurs when one of the $2k$ dividers passes through two points of $\hat{X}$. To count the number of 'dichromatic' events we first show

Lemma 2. Let p be an arbitrary point, which does not coincide with a red point and is not colinear with two red points. The number of times, m, during the algorithm when p crosses any of the k rotating dividers of the red points is $O(k)$.

Proof. Consider L_r, the $r+1$-divider (for some r) of the red points. Let q_i $(i = 1, 2)$ be the pivots of L_r during any two crossings of p over L_r. If the two crossings are consecutive in time (ie. there are no other crossings during the interval between these two crossings) then we claim that L_r is oriented from q_1 to p (during the first crossing) iff L_r is oriented from p to q_2 (during the second crossing) (see [ELSS] for a similar result). Let Q_r be the set of points q such that q is the pivot of the rotating $r+1$-divider of the red points during some instance when p crosses L_r. Let $Q = \bigcup_{r=0}^{k-1} Q_r$. Thus $|Q| \leq m$, and as each $q \in Q$ is a pivot for just one crossing, $|Q| = m$. For any line L through p, the above claim shows that at least $\frac{1}{2}\max\{0, Q_r - 3\}$ points of Q_r lie on each side of L. Thus at least $\max\{0, \frac{m-3k}{2}\}$ points of Q lie on each side of L. Choosing L to be L_r for some r, we see that there are at most r points strictly to the right of L. Thus $k - 1 \geq r \geq \frac{m-3k}{2}$, or $m < 5k$. □

Each dichromatic event corresponding to a red divider crossing a black point can be charged to the black point; conversely when a black divider crosses a red point, the red point is charged. The number of charges to each point is $O(k)$ by the lemma. Thus there are $O(nk)$ dichromatic events. A simple modification of the above lemma to allow p to be one of the red points leads to the following:

Corollary. There are $O(nk)$ monochromatic events.

We provide another view of the corollary which is of independent interest. Recall $N_k(n) = \max_{|G|=n} N_k(G)$, where $N_k(G)$ is the number of k-sets for the point arrangement G. We define $\bar{N}_k(G) = \sum_{i=0}^{k-1} N_i(G)$, and $\bar{N}_k(n) = \max_{|G|=n} \bar{N}_k(G)$. Then we have shown $\bar{N}_k(G) = O(kn)$.

Thus the total number of events (monochromatic and dichromatic) is $O(nk)$. Since each event incurs $O(\log^2 n \log k)$ work, we conclude that finding all special k-dividers of $\hat{X}$ takes $O(nk\log^2 n \log k)$ time.

If k is $\omega(\frac{n}{\log n})$, it is slightly more efficient to first preprocess the rotational order of $\hat{X}$ about each point p in $\hat{X}$, using time $O(n^2 \log n)$. With this preprocessing, each event can be processed in $O(\log n)$ time and so the rotation takes a total of $O(nk\log n)$ time. We have now shown:

Theorem 4. The special k-dividers of a bicolored set of n points in the plane can be computed in time $O(n\log n \cdot \min\{n, k\log k \log n\})$.

In 3-dimensions, the k-hull of a set X of points is defined to be the set of points p such that any plane through p has at least k points on each side of the closed half-space determined by the plane. Again by Helly's theorem [YB], we know that the k-hull is non-empty if $k \leq \lceil n/4 \rceil - 1$. Again we call the k-hull in this (extremal) case the *center* of the set of points. We want a characterization theorem similar to theorem 1 but the generalization of special k-dividers is not obvious. Since we do not have an algorithm similar to the one above, we will first give an alternative proof of theorem 1 which can be generalized.

The non-trivial direction of theorem 1 is to show that the intersection H_k of the special k-half-planes is contained in the k-hull. Let p be any point not in the k-hull and L a line through p such that there are less than k points of X in the (closed) positive half-plane L_+ to the right of L. To show that p is not in H_k, translate L in the negative direction until a line M parallel to L is reached such that there are k points in the positive half-plane M_+. Let C_1 (resp. C_2) be the convex hull of those points of X in M_+ (resp. complement of M_+). Consider the two co-tangents of C_1 and C_2 which separate them. They are special k-dividers. It is easily seen that for at least one of the special half-planes determined by these special dividers, p is not in the special half-plane -- proving that p is not in H_k.

We are now ready to generalize the above proof. Planes are assumed to be oriented and the points of X are *non−degenerate* in the sense that no 4 points are co-planar. We define a *k−dividing plane* (or k-divider) to be a plane with at most $k-1$ points of X in the (open) positive side of the plane and at most $n-k$ points of X on the (open) negative side. Again, note that each orientation in space determines a unique k-divider.

The k-divider is *special* if it contains 3 points of X and is at the same time a $k+1$-divider. Note that a special k-divider has either $k-1$ or $k-2$ points of X on the open positive side. The closed half-space on the negative side of a special k-divider is called a *special $k-half-space$*.

Theorem 5. The k-hull is the intersection H_k of all the special k-half-spaces.

Proof. It is easy to see that the k-hull is contained in H_k. Conversely, let p be a point not in the k-hull and P a plane through p with less than k points of X on, say, the closed positive side of P. We show that p is not in H_k. Translate P to the parallel plane Q with exactly k points in its closed positive side. Form the convex hulls of the two subsets of X determined by Q (with k and $n-k$ points, respectively). Now consider the set of 'separating planes' P (ie. the convex hulls lie on opposite sides of P) where P is a supporting plane for both convex hulls and is also a special k-divider. It is not hard to see that p is not in at least one of the special half-spaces determined by this set of special dividers. □

In 3-d, there is no obvious 'rotating plane' algorithm generalizing the algorithm in 2-d. To simplify the problem, we fix an arbitrary point $p_0 \in X$ and consider rotating a plane P such that it continuously contains p_0. Without loss of generality, assume p_0 is the only point of X which lies on the $z = 1$ plane. We now reduce to the '2.5 dimensional' algorithm.

Using p_0 as the center of projection, the rest of the points are projected onto the $z = 0$ plane. The projected points on the $z = 0$ plane are colored "black" (resp. "red") if the original point lies above (resp. below) the $z = 1$ plane. Let $\hat{X}$ denote this set of (bicolored) projected points. Similarly, any plane P through p_0 is projected onto the $z = 0$ plane as a straight line $L = L(P)$. Then a plane P is a special k-divider if and only if $L(P)$ is a special k-divider of $\hat{X}$.

To find the special k-half-spaces of a set of n points in 3-dimensions, we find the special k-dividers for the projected problem using p as the center of projection, for each $p \in X$. From these we obtain the corresponding special k-dividers in 3 dimensions. Since there are $O(n^3)$ special k-half-spaces, their intersection can be found by a divide-and-conquer method in time $O(n^3 \log n)$; here we rely on an algorithm of Preparata and Muller [PM] for intersecting two polyhedra in time $O(m \log m)$ where m is the total number of faces. This proves

Theorem 6. The k-hull of n points in 3-dimensions can be computed in time $O(n^2 \log n \cdot \min\{n, k \log k \log n\})$. In particular, the center (ie. the $(\lceil n/4 \rceil - 1)$-hull) can be found in $O(n^3 \log n)$ time.

3. On Megiddo's Technique: General Remarks

In the next two sections we illustrate the 'parametric' searching technique introduced by Megiddo [M]. Our use of his technique is quite striking because (1) it is not apparent that our problems are parametrized and (2) in some of our algorithms, we have to apply his technique recursively in a way that is quite non-trivial. In order to be able to use his technique we shall need efficient parallel sorting algorithms, in particular, either the n-processor, $(\log n)$-depth network of [AKS], or the $O(n \log n)$-processor, $O(\log n)$ parallel time algorithm of Preparata [P].

In general terms, his technique provides a way to search a partially ordered space, of polynomial size, without actually constructing the space (the polynomial may be large). Instead, an *implicit* binary search of the space is made. We use a sorting algorithm to guide this search (Megiddo's technique is not restricted to sorting algorithms). Typically, a sorted order will correspond to a region of the space being searched. So resolving a comparison in the sort corresponds to reducing the size of the space in which a solution is known to lie. As might be expected, a single comparison is expensive; the surprising observation is that several comparisons can be batched relatively cheaply. This leads us to use a parallel sorting algorithm, for we (in our serial algorithm) can batch the comparisons that are carried out simultaneously by the parallel algorithm. A fascinating feature of this implicit search is that it can be applied recursively, each level of the recursion adding some poly-logarithmic factor to the complexity. These remarks are most easily understood by considering some examples, such as the problems we solve below.

4. Finding a Ham Sandwich Cut

Instead of computing the entire k-hull, some applications require just a single point in it. For $k \leq n/4$, we can reduce this simpler problem to the following: given X, a set with n red points and n black points in the plane, find a line (called a 'fair cut') which simultaneously separates each of the monochromatic subsets evenly. The existence of a suitable line is guaranteed by the Ham Sandwich theorem. It is easy to see that each fair cut can be associated with some line through two points of X. By checking each pair of points in turn, the desired cut can be found in $O(n^3)$ time. We shall drastically reduce this bound.

To simplify the discussion, we may assume that n is odd and no three points are colinear. A *halver* of a set Y of n points (recall n is odd) is an (oriented) line with at most $\frac{n-1}{2}$ points of Y in each of the open half-planes it determines. (Thus a halver is just a $\frac{n+1}{2}$-divider as defined in the last section.) For each orien-

tation θ, let L_θ^R be the halver of the red subset of X with orientation θ. Similarly L_θ^B is the halver of the black subset. Let $r(\theta)$ denote the distance that L_θ^B lies to the right of L_θ^R. This distance is negative (resp. zero) if L_θ^B actually lies to the left of (resp. coincides with) L_θ^R. Thus, our goal is to find some (any) orientation θ^* such that $r(\theta^*) = 0$. Note that a fair cut is easily derived from θ^*, in linear time. We note $r(\theta)$ varies continuously with θ.

For any point p and orientation θ, let $p(\theta)$ denote the projection of p along the direction θ onto the x-axis. For the orientation θ^*, let p^* be used instead of $p(\theta^*)$. Although we do not know the value of θ^*, we shall show how we can sort the points in $X^* = \{p^* : p \in X\}$ (which will allow us to deduce the value of θ^*). Let us inductively suppose that an angular interval $\Phi \subseteq [0, \pi]$ is known such that either Φ is a single angle θ with the property $r(\theta) = 0$, or Φ is an open interval (θ_1, θ_2) such that $r(\theta_1) > 0$ and $r(\theta_2) < 0$. (And thus there is a value $\theta^* \in \Phi$, with $r(\theta^*) = 0$.) Initially, we may take Φ to be $(0, \pi)$ unless $r(0) = 0$ in which case we let $\Phi = [0, 0]$. Clearly Φ contains some θ with $r(\theta) = 0$. Suppose that in sorting X^*, we compare $p_i^* : p_j^*$, where $p_i, p_j \in X$. Let $\theta_{i,j}$ denote the angle of the line from p_i to p_j. Assume $0 \le \theta_{i,j} < \pi$ (otherwise, take $\theta_{j,i}$ instead). If $\theta_{i,j} < \theta_1$ or $\theta_2 < \theta_{i,j}$ then the relative ordering of p_i^* and p_j^* can be deduced at once. If $r(\theta_{i,j}) = 0$ then θ^* is determined. Otherwise let $r(\theta_{i,j}) > 0$ (the case $r(\theta_{i,j}) < 0$ is similar). We refine Φ into $\Phi' = (\theta_{i,j}, \theta_2)$; since the relative ordering of $p_i(\theta)$ and $p_j(\theta)$ is fixed for $\theta > \theta_{i,j}$, and hence for $\theta \in \Phi'$, and as there is a value of $\theta^* \in \Phi'$, we can deduce the relative ordering of p_i^* and p_j^*. Clearly, for any θ, $r(\theta)$ (and hence the comparison $p_i^* : p_j^*$) can be computed in linear time (using a linear time median algorithm [AHU]). It is easy to see that this gives an $O(n^2 \log n)$ time algorithm for computing θ^*. To obtain a faster algorithm, we apply Megiddo's technique.

We use a p processor, h time parallel sorting algorithm to sort X^*. Assume inductively that an angular interval Φ_k ($k = 0, 1, \ldots, h$) containing (some) θ^* is known before the start of the $(k+1)$st step, where Φ_k is either an open interval or a single orientation. In the $(k+1)$st parallel step of the sorting algorithm, p comparisons of the form $p_i^* : p_j^*$ are made. The p comparisons determine p orientations $\theta_{i,j}$. As above, we may assume that $\theta_{i,j}$ is in the range $(0, \pi)$. For each $\theta_{i,j}$ lying outside the range Φ_k, the outcome of the corresponding comparison is known, as shown in the previous paragraph. For those orientations in Φ_k we first sort them (using any straightforward method) and then do a 'binary search' of the sorted orientations: each probe of the binary search computes $r(\theta_{i,j})$ for some $\theta_{i,j}$ in Φ_k. Unless $r(\theta_{i,j}) = 0$ (in which case we are done), we refine the interval Φ_k as described in the last paragraph. Each probe takes $O(n)$ time, and $O(\log p)$ probes suffice to determine an angular interval $\Phi_{k+1} \subseteq \Phi_k$ containing θ^* with the property that each of the p comparisons in the parallel step is decided. Therefore each parallel step is accomplished in $O(n \log p + p \log p)$ serial steps. (This can be reduced to $O(n \log p + p)$ serial steps, as follows. We do not have to sort the orientations in Φ_k; it suffices to find the orientations probed by the binary search, for which we use a fast median algorithm [AHU]. Each step of the binary search reduces by half the size of the set to be searched, so we only need $O(p)$ serial steps to find the orientations to be probed.) Since there are h parallel steps, the total time is $O((n \log p + p)h)$. If one uses the sorting algorithm of Preparata with $p = O(n \log n)$, and $h = O(\log n)$, or the asymptotically more efficient network of [AKS], the total time complexity becomes $O(n \log^2 n)$. For reference, call the serial algorithm just described the *derived* algorithm (with respect to any parallel sorting algorithm).

We now prove the correctness of the derived algorithm. For any orientation θ in $(0, \pi)$, let the θ-*ordering* of X refer to the total order on X induced by the ordering of the projection of X onto the x-axis in the direction θ. (This ordering is well-defined because the orientation of the x-axis is 0.) Consider any comparison $p_i^* : p_j^*$ made in the kth parallel step of the parallel sorting algorithm. The outcome of this comparison depends on the relative order of p_i and p_j in the θ^*-ordering. The relative order of p_i and p_j in the θ-ordering is invariant as θ varies in Φ_{k+1}. And this is the outcome we have assigned to the comparison $p_i^* : p_j^*$. Next observe that

$$(0, \pi) = \Phi_0 \supseteq \Phi_1 \supseteq \cdots \supseteq \Phi_m, \ (m \le \log n). \qquad (*)$$

Let T be the θ-ordering on X for some θ in Φ_m. We conlude from $(*)$ that the outcomes of all the comparisons obtained by the (parallel or derived) algorithm are consistent with T; the correctness of the parallel algorithm then implies that the outcomes determine a unique total ordering T' of X; in fact $T' = T$. We can say more:

Correctness Lemma. The interval Φ_m consists of a single orientation θ^* with $r(\theta^*) = 0$.

Proof. Suppose $\Phi_m = (\theta_1, \theta_2)$, $\theta_1 < \theta_2$. Then $r(\theta_1) > 0$, $r(\theta_2) < 0$. Notice that $r(\theta)$ varies continuously with θ. Thus there is some value θ^* such that $r(\theta^*) = 0$; then for $\theta = \theta^*$ the dividers of the two sets are coincident and thus pass through a red point p and a black point q. So the relative order of $p(\theta)$ and $q(\theta)$ depends on whether $\theta < \theta^*$ or $\theta > \theta^*$ and thus is not constant over Φ_m. Contradiction. Thus Φ_m is a single

orientation. □

We prove this lemma under even weaker assumptions below.

An improved algorithm derived from parallel median algorithms. As noted above the parallel algorithm used for sorting need not be from the 'Network Model'; the more general 'adaptive' model of Valiant [V] can be used. Better still, we can use a parallel median rather than a parallel sorting algorithm. These remarks open the way for even faster algorithms since a faster than $O(\log n)$ time median algorithm is possible in Valiant's model (this is not so in the network model). Indeed, [CY] describes an $M(n) = O((\log\log n)^2)$ time algorithm for finding the median with n processors. The method for deriving an algorithm (for finding a fair cut) from parallel median algorithms is exactly the same as above: at the beginning of the kth parallel step, we inductively assume that we have an angular interval Φ_k such that (i) either Φ_k is an open interval (θ_1, θ_2) with $r(\theta_1) > 0 > r(\theta_2)$ or Φ_k consists of a single orientation θ with $r(\theta) = 0$, (ii) for all comparisons $p_i^* : p_j^*$ in the $(k-1)$st step, the relative order of p_i and p_j in the θ-ordering is invariant as θ varies over Φ_k, and (iii) $\Phi_k \supseteq \Phi_{k+1}$, $k = 0, 1, \ldots, m \leq M(n)$. The correctness of the parallel median algorithm implies that the derived algorithm will find a unique element p^* in X such that the median of X under the θ-ordering is invariant as θ varies over Φ_m. (Actually, it finds two median elements, for there will be a tie for median at angle θ^*.) We can still prove the Correctness Lemma; the only change to the above proof is to note the points p^* and q^* (in the proof of the correctness lemma) are the two medians. This results in:

Theorem 7. A fair cut for the ham sandwich problem can be found in time $O(n\log n M(n)) = O(n\log n(\log\log n)^2)$ time where $M(n)$ is the parallel time to find the median of n elements using n processors in the Valiant model.

If we make the further assumption that there is a unique θ^* we can extend Megiddo's technique to obtain even faster algorithms which are probabilistic.

Our approach is based on the following idea. We choose a random point p, which with probability 1/4 lies between the $n/2+1$st and the $3n/4$th point in the θ^*-ordering. We check this by finding two sets of points; one consisting of at least $n/2$ points guaranteed to be before p in the θ^*-ordering, the other (denoted S) of at least $n/4$ points guaranteed to be after p in the θ^*-ordering. If we do not find these sets we choose another random point p. The set S cannot include the median point, for there are at least $n/2$ points before p and hence there are at least $n/2+1$ points before any point in S. So we can restrict our search to the remaining at most $3n/4$ points; however this is no longer a median problem, so we are forced to consider a slightly more general problem.

We give an algorithm to find a cut with r red points and b black points to its left, the cut passing through one red and one black point. We define L_θ^R to be the line at orientation θ, passing through a point in the red set and having r red points strictly to its left; similarly, L_θ^B passes through a black point and has b black points strictly to its left. $r(\theta)$ is the distance that L_θ^B lies to the right of L_θ^R. Our goal is to find an angle θ^* such that $r(\theta^*) = 0$. As above, we assume that $r(0) > 0$, and $r(\pi) < 0$. In general, such a cut need not exist. But in the instances we consider, a cut always exists.

Our algorithm follows the intuitive outline given above. Suppose we have n points altogether, let $k = r + b + 1$, and without loss of generality suppose $k \leq \lceil n/2 \rceil$. The cut passes through the kth and the $k+1$st points in the θ^*-ordering, which we shall find. For simplicity, suppose n is even (the changes to be made for n odd are straightforward). We select a random point p; with probability 1/16 it will lie in the range $(10n/16, 11n/16]$. We confirm that p lies in the range $(n/2, 3n/4]$, as follows. We compare all the other points to p and seek to show that at least $n/2$ of them precede p, and at least $n/4$ follow p, in the θ^*-ordering. On comparing these $n-1$ points with p we obtain $n-1$ angles $\theta_1 \leq \ldots \leq \theta_{n-1}$, say. By performing 4 comparisons (the first 4 in a binary search) we either determine θ^*, or find an i such that $\theta^* \in (\theta_i, \theta_{i+n/16})$. In the latter case we have determined the relative order of p and $15n/16$ of the points. If p is in the range $(10n/16, 11n/16]$ then at least $15n/16 - 6n/16 > n/2$ of the points are shown to precede p; likewise, at least $15n/16 - 11n/16 = n/4$ are shown to follow p. If we have not shown that p is preceded by at least $n/2$ points and followed by at least $n/4$ points, we randomly select another p. The work in this step takes expected time $O(n)$ (note we do not have to sort the angles θ_i).

Either we compute a θ for which $r(\theta) = 0$ or we find a p for which we show that it is preceded by at least $n/2$ points and followed by at least $n/4$ points (denoted S), in expected time $O(n)$. Since $k \leq n/2$ we deduce that the points in S cannot include the kth or the $k+1$st points; so points in S can be discarded. We recursively solve the problem on the remaining set of at most $3n/4$ points, using the same values of r and b. (If k had been greater than $n/2$ we would have to reduce the values of r and b by the number of points of each color removed.) We note the new problem has a solution if the old problem had a solution. Since we start by seeking the ham sandwich cut, which

is known to exist, we are guaranteed that each recursive problem we generate has a solution. Notice also that if the old problem had a unique solution then so does the new problem (if we restrict the solution to the new problem to the angular range $(\theta_i, \theta_{i+\pi/16})$, in which range we have determined θ^* lies). We obtain:

Theorem 8. A fair cut can be found in linear time using a Las Vegas type algorithm, if the cut is unique.

5. Finding a point in the center

The algorithm in the last section easily finds a point in the $n/4$-hull (since it can be used to find a 1/4-partitioner, section 6(1)). Unfortunately (and somewhat surprisingly) it does not appear to extend directly to finding a point in the center (i.e. the $n/3$-hull). We now develop such an algorithm. Megiddo's technique will again be exploited (to the hilt).

First we note that it is easy to determine if a given point p is in the center: sort the points of X in angular order about p and then rotate a line about p through a full circle of angles, keeping count of the number of points on the strictly left side of the rotating line. The point p is in the center if the number of points on the left is always at least $\lceil n/3 \rceil - 1$. The time complexity of this procedure is $O(n\log n)$. There are two noteworthy points. First, to define a sorted order we have to start it somewhere -- say at the angle 0. Second, we note that if one point lies at angle $\pi+\alpha$ and a second one at angie β, $\alpha, \beta < \pi$, we are really interested in the relative size of α and β when performing the rotation. That is, we want to know the rotational order of the points when those points below the horizontal line through p are reflected through p; henceforth we shall use the term rotational order for this order.

If we know a point p^* in the center, then the rotational order T^* of the points in X about p^* is easily determined. We show how to compute (some) T^* without knowing (any such) p^*, using a p-processor parallel sorting algorithm. What does 'comparing' a pair of points p_i and p_j in X mean, relative to the ordering T^*? The relative rotational order of p_i and p_j about a point q depends on which of 6 regions q lies in (see figure 3). The regions are defined by the line L_{ij} through p_i and p_j, and by the horizontal lines through p_i and p_j. So to resolve the comparison we determine if the center intersects any of these lines and if not in which region the center lies. To do this we use an algorithm which given a line L determines if the center intersects L, and if so gives a point on L in the center, and if not determines on which side of L the center lies. This algorithm runs in time $O(n\log^3 n)$. The idea is similar to the ham sandwich cut algorithm; we describe it following the main algorithm.

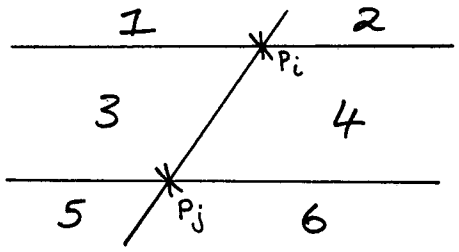

figure 3.

To eliminate the consideration of the horizontal lines when performing comparisons between pairs of points we first either determine between which pair of horizontal lines the center lies, or we find a point in the center on one of these lines. We do this by performing a binary search of the n horizontal lines, using the algorithm mentioned in the last paragraph; we use a total of $O(n\log^4 n)$ time.

(1) The next crucial idea is how to 'batch process' p of these comparisons between pairs p_i and p_j, performed during one parallel step of the sorting.

The p comparisons give rise to p lines of the form $L_{i,j}$, which determine $O(p^2)$ regions. Within any region the outcome of these comparisons is constant. The problem reduces to finding out which of these regions contain p^*. By drawing horizontal lines through the $O(p^2)$ intersections of pairs of these lines, we obtain horizontal slabs where within each slab, the original lines $L_{i,j}$ are totally ordered. So if we only knew the slab S^* which contains the center, we could then sort the lines within S^*, and perform a binary search to find a region containing p^*. Again, we show how to sort the lines $L_{i,j}$ without knowing S^* in advance. What does comparing (ordering) a pair of lines $L_{i,j}$ and $L_{i',j'}$ mean? It reduces to knowing whether the center lies above or below the horizontal line through their intersection point. This being the case, we now use a q processor parallel sorting algorithm to sort the $L_{i,j}$'s within S^*. Each parallel step of this last algorithm makes q comparisons and we are led to the next issue:

(2) How to 'batch' q of these 'comparisons' between $L_{i,j}$ and $L_{i',j'}$.

This is not so difficult because we can indeed do a binary search among the q horizontal lines induced by the comparisons. Since each step of the binary search takes $O(n\log^3 n)$ time, we can perform step (2) in time $O(n\log^3 n\log q + q)$ time. If we use Preparata's parallel sorting algorithm (or the [AKS] network) we can perform step 1 in time of $O(n\log^3 n\log^2 p + p\log^2 p)$. So to find the rotational order of the points about p^* (using Preparata's

algorithm or the [AKS] network) takes time $O(n\log^6 n)$. At the end of this process we have either found a point p^*, or a bounded region R, with respect to which the rotational order of the points in X is fixed. The region R is the intersection of $O(\log n)$ regions, one for each parallel step of the point sorting algorithm. Such a region is bounded by four lines: two horizontal lines bounding the slab in which the center lies, and the two lines L_{ij} between which the center lies.

Correctness Lemma. The center contains R.

Proof. From the existence of the center and the form of the algorithm it is clear that R contains a point in the center. Because the rotational order of the points is the same about any point in R, either every point in R is in the center or none of R is in the center, and plainly only the first case is possible. □

It remains to give an algorithm to determine whether the center intersects a given line L, or on which side of L it lies. For each orientation θ and point x, let $l_\theta(x)$ denote the line through x and with orientation θ. Define $f_\theta(x)$ to be the fraction of the points of X which lie on the closed half-plane to the right of $l_\theta(x)$ and also define $g(x) = \min_\theta f_\theta(x)$, $g_1(x) = \min_{0<\theta<\pi} f_\theta(x)$, and $g_2(x) = \min_{-\pi<\theta<0} f_\theta(x)$. We state some simple properties of f, g, g_1, and g_2.

Lemma 4.

(i) Let L be any line (imagined as the x-axis). Then for each θ, the function $f_\theta(x)$ is monotone as x varies along L. It is constant for $\theta = 0$ or π, decreasing from 1 to 0 for $\theta \in (0, \pi)$ (the 'upward' orientations), and increasing from 0 to 1 for $\theta \in (-\pi, 0)$ (the 'downward' orientations).

(ii) $g_1(x)$ is a decreasing function on L and $g_2(x)$ is an increasing function on L. Let I be the interval over which $g_1(x) = g_2(x)$.

(iii) For any x, $g(x)$ attains its minimum at some set $\Theta(x)$ of orientations. Suppose x is not in the center. If $\Theta(x)$ consists only of upward (resp. downward) orientations then x lies to the left (resp. right) of I. Otherwise $\Theta(x)$ has both types of orientations and x lies in I.

We seek a point p^* in I. To guide the search for p^* we find the rotational order of the points in X about p^*, using a p processor algorithm, running in parallel time h. A comparison of two points p_i and p_j, as described above, determines a line L_{ij}; it intersects L in a point, x say. The result of the comparison depends solely on which side of x the point p^* lies. We determine whether I includes x (in which case we are done), or if I is to the left or right of x, by rotating a line about x and computing $f_\theta(x)$ as θ varies; using (iii) above we deduce where x lies. This takes time $O(n\log n)$. To batch p of these comparisons, we sort the resulting p points on L and perform a binary search among them (actually, find medians etc.). This yields an interval J (containing p^*) over which the comparisons performed so far have a fixed result. This takes time $O(n\log n\log p + p)$. Hence to find the rotational order of the points about p^* takes time $O((n\log n\log p + p)h)$. Using either Preparata's algorithm or the [AKS] network this becomes a time of $O(n\log^3 n)$.

Correctness Lemma. The algorithm either finds a point in the center, a point in I, or an open interval J with respect to which the rotational order of the points in X is fixed, and with $J \subset I$, the L-maximal center.

By the correctness lemma if the procedure does not find a center, it finds a point in I. Let x be this point. As x is not in the center, $g(x) < 1/3$. Observe that this implies that there exist two orientations, an upward θ_1 and a downward θ_2, such that $f_{\theta_1}(x) = f_{\theta_2}(x) < 1/3$. See figure 4.

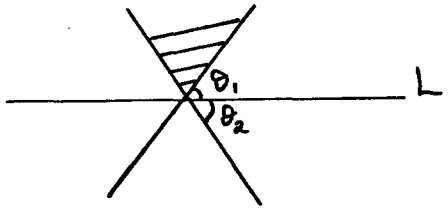

figure 4.

It follows that the center of S must lie in the shaded area of the figure (the side of L that this shaded area lies on depends on θ_i), because for any point not in this region either the line with orientation θ_1, or the one with orientation θ_2 is guaranteed to have fewer than $n/3$ points to its right. So if L does not pass through the center of X, the procedure for computing a point in the L-maximal center also tells us on which side of L the actual center of X lies.

A much more complex version of these ideas gives us an algorithm for determining a point in the center in 3-dimensions. These results are summarized in

Theorem 9. A point in the center of n points can be found in time $O(n\log^6 n)$ (in 2-dim) and $O(n^2\log^{10} n)$ (in 3-dim).

The method probably can be extended to any dimension.

6. Applications

1) Partitioners. An algorithm for finding a 1/4-partitioner is easily obtained using the cake-cutting algorithm: given a set of points X, we find a line which partitions it into two equal halves. By appeal to the cake-cutting algorithm, we find another line such that the two lines form a 1/4-partitioner, in deterministic time $O(n\log n(\log\log n)^2)$. We can also use the probabilistic algorithm here, obtaining a running time of $O(n)$. For let L be the first partitioner, and the black (resp. red) points be those above (resp. below) L. We notice that as we increase θ, the intersection of L and L_θ^B moves left to right along L, while the intersection of L and L_{θ_R} moves right to left. Hence there is a unique interval (in fact a single point) of values of θ for which $L_\theta^B = L_\theta^R$, and hence for which $r(\theta)=0$. [We are indebted to L. Guibas (in conversation) for posing the Ham Sandwich cut problem, since our original interest was in finding partitioners.]

2) Polygon Retrieval. Using the algorithm in (1), the preprocessing time for the polygon retrieval algorithms in [W,EW3] is reduced from $O(N_{n^2}(n)\log^2 n)$ to $O(n\log^2 n(\log\log n)^2)$, or if we use a probalistic algorithm to $O(n\log n)$.

3) Intersecting Lines and Points. Hopcroft posed the following problem which arose in the context of collision detection in robotics: given Y a set of n lines and X a set of n points, determine if any point lies in any line. The naive solution takes $O(n^2)$ time. We first sketch a solution (this was also known to Hopcroft and Seidel) which takes time $O(n^{3/2}\log n)$. Based on this solution, we shall obtain further improvements using 1/4-partitioners.

To see the first solution, we arbitrarily divide X into $n^{1/2}$ groups each with $n^{1/2}$ points. It is enough to show how to check if any line in Y intersects any point in one of these groups in time $O(n\log n)$: The pairs of points in a group determine $O(n)$ directions which we combine with the n directions of the lines and then sort. (In fact, it is sufficient to sort the directions determined by pairs of points, obtaining an order S, and then by binary search insert the directions of the lines into S, without requiring that these directions be sorted with respect to each other -- this remark is pertinent to the generalization below.) Project the $n^{1/2}$ points along any direction, say θ_0, determined by a pair of points and store these projected points in a balanced binary tree. Now process the sorted sequence of directions as follows. Beginning with θ_0, if we encounter a direction θ determined by a pair of points, we transform the binary tree so that the sequence of points it stores represent a projection along θ: it is not hard to see that this (essentially) involves a deletion from and an insertion into the tree. If the θ we encounter is the direction of a line, we can in $O(\log n)$ time check if the line contains any points on the group. Since each of the $O(n)$ directions can be processed in $O(\log n)$ time, the total time to process one group is $O(n\log n)$, and thus the time to process all the groups is $O(n^{1.5}\log n)$.

More generally, let $T(n,m)$ denote the complexity of the problem with an input set X of n points and an input set Y of m lines. By duality, we have $T(n,m) = T(m,n)$. It is easy to use the above technique to show that for $n \le m^2$,

$$T(m,n) = T(n,m) = \begin{cases} m\log n & \text{if } n \le m^{1/2} \\ nm^{1/2}\log n & \text{if } m^{1/2} < n \le m^2 \end{cases} \quad (3)$$

We now improve the just derived bound of $T(n,n) = O(n^{3/2}\log n)$. The basic idea is illustrated as follows: suppose we have to check the intersection of a set Y of lines with a set X of points. Let X_i ($i = 1, \ldots, 4$) be subsets of X determined by a 1/4-partition. Note that any line intersects at most 3 of the quadrants of the 1/4-partition. The original problem is thus reduced to four subproblems of checking intersections of subsets $Y_i \subseteq Y$ with X_i ($i = 1, \ldots, 4$), where Y_i consists of the lines which intersect the quadrant containing X_i. Furthermore, $\sum_{i=1}^{4} |Y_i| \le 3|Y|$. The recursive application of this idea will obtain $T(n,n) = O(n^c\log n)$ for some $c < 1.5$. To get the best constant for c, we recall Willard's [W] J-way division of a set of points (a 1/4-partition may be regarded as a J-way division with $J = 2$). The pertinent property is that a J-way division consists of J lines (or half-lines) which partition X into $2J$ approximately equal parts such that an arbitrary line intersects at most $J + 1$ of these parts. Furthermore, such a division can be obtained by $J-1$ applications of the partition algorithm in section 6. We chose $J = 3$ to obtain the best constant in the following construction.

We first set up a 'search tree' on X in which each node has degree 6. The root is associated with the entire set X and a 3-way partition of X. In general, if a node u is associated with a subset X' of X and a 3-way division of X' then each child of u is recursively associated with one of the 6 subsets formed by the 3-way division of X'. The height of the search tree is taken to be $\log_{24} n$ and thus each leaf is associated with a subset of size $O(n^{\log_{24} 4})$. The cost of forming the 3-way division at all the nodes of the search tree is seen to be $O(n\log^2 n(\log\log n)^2)$, relying on the $O(n\log n(\log\log n)^2)$ algorithm for partitioning given in section 6.

The 'search' for intersections of lines in Y with points in X proceeds as follows: initially, we are at the root and want to check if any line in Y contains any point in X. In general, suppose we are at some node u and want to check if any line in some set $Y' \subseteq Y$ contains any point in the set $X' \subseteq X$ associated with u. If $|Y'| \leq |X'|^{1/2}$ then we solve this problem directly by (3) in time $O(|X'|\log|X'|)$. Similarly, if $|Y'| \geq |X'|^2$ we solve it directly in time $O(|Y'|\log|Y'|)$. If u is a leaf then $|X'| = O(n^{\log_{24}4})$ and we also solve this problem directly by (3) in time $O(n^{\log_{24}4}|Y'|^{1/2}\log n)$. Otherwise, for each line in Y' and for each of the 6 quadrants determined by the 3-way division at u, we check if the line intersects the quadrant. Note that the line intersects at most 4 quadrants. The problem is now reduced to 6 subproblems where we want to check for each quadrant whether any line which intersects that quadrant contains any of the points in that quadrant. Each of the subproblems is naturally solved recursively at the children of u.

We now assess the complexity of this procedure. Let U be the set of nodes in the search tree which were visited. For $u \in U$, let n_u (resp. m_u) denote the size of the set X' (resp. Y') of points (resp. lines) to be checked at u. We may partition U into four classes: U_0 (resp. U_1) consists of the nodes u where $n_u^{1/2} \geq m_u$ (resp. $n_u^2 \leq m_u$), U_2 consists of the leaves (of the search tree) which were not counted in $U_0 \cup U_1$, and U_3 consists of the rest of U (i.e. those nodes which spawn six subproblems). The work done at nodes in U_0 is seen to be

$$\sum_{u \in U_0} n_u \log n_u = O(n\log n),$$

since $\sum_{u \in U_0} n_u = O(n)$. The work at U_1 is

$$\sum_{u \in U_1} m_u \log m_u = O(mn^{\log_{24}4}\log n),$$

since $\sum_{u \in U_1} m_u = O(mn^{\log_{24}4})$. The work done at U_2 is

$$n^{\log_{24}4}\log n\left(\sum_{u \in U_2} m_u^{1/2}\right) = n^{\log_{24}4}\log n \cdot (mn)^{1/2}.$$

To see the above, note that $|U_2| = O(n^{\log_{24}4})$, $\sum_{u \in U_2} m_u = O(mn^{\log_{24}4})$ and the sum $\sum_{u \in U_2} m_2^{1/2}$ is maximized when all the m_u are equal. Finally, the work done at a node $u \in U_3$ is $O(m_u)$. Summing over all such nodes, we have

$$\sum_{u \in U_3} m_u \leq m \cdot \sum_{i=0}^{\log_{24} n} 4^i = O(mn^{\log_{24}4}).$$

The total cost (including the 'preprocessing cost' of computing the search tree) from the above analysis is summarized in:

Theorem 10. $T(n,m) = O(n\log^2 n(\log\log n)^2 + mn^{\log_{24}4}\log n + m^{1/2}n^{1/2+\log_{24}4}\log n)$. In particular, $T(n,n) = O(n^{1+\log_{24}4}\log n) = O(n^{1.437}\log n)$.

7. Conclusions.

The general contribution of this paper is to introduce a natural generalization of the concepts of convex hulls and centers, and to investigate a number of computational problems surrounding it. The rich connection between k-hulls and the previously studied concepts of k-belts, k-sets, α-partitioners, and the classical Ham Sandwich theorem is particularly noteworthy, and no doubt sheds some insights into these concepts.

The algorithms presented here develop, or at least make more accessible, some powerful algorithmic techniques which are quite new in computational geometry. Specifically, we have extended the domain of applications for Megiddo's technique. We expect to find further new applications for these techniques.

Finally, our 'parametric searching' algorithms are significantly faster than previous solutions. This in turn translates into efficient algorithms for a number of applications. Again, we expect other applications to be found.

Acknowledgements

We are indebted to R. Pollack for references [L,ELSS] and for pointing out that our algorithm in section 2 had previously been discovered by Lovász.

References

[Ag] M. K. Agoston, *Algebraic Topology*, Marcel Dekker, New York, 1976.

[AHU] Aho, Hopcroft, and Ullman, The Design and Analysis of Computer Algorithms,Addison Wesley, 1974.

[AKS] M. Ajtai, J. Komlós and E. Szemerédi, "An $O(n\log n)$ Sorting Network", *15th STOC*, 1983, 1-9.

[C] B. Chazelle, "Optimal algorithms for computing depths and layers", Brown University Technical Report No. CS-83-13, March 1983.

[CY] R. Cole and C. K. Yap, "A parallel median algorithm", preliminary version, September 1983.

[DE] D. Dobkin and H. Edelsbrunner, private communication.

[DS] L. Dubins and E. Spanier, "How to cut a cake fairly", *AMM*, 68(1961), 1-17.

[EW1] H. Edelsbrunner and E. Welzl, "On the number of line-separations of a finite set in the plane", to appear, *J. Comb. Theory, A*.

[EW2] H. Edelsbrunner and E. Welzl, "Halfplanar Range Estimation", Report F98, Inst. for Inf. Proc., Tech. Univ. of Graz, Austria (1982).

[EW3] H. Edelsbrunner and E. Welzl, "Halfplanar range search in linear space and $O(n^{0.695})$ query time",

Report F111, Inst. for Inf. Proc., Tech. Univ. of Graz, Austria (1983).

[ELSS] P. Erdős, L. Lovász, A. Simmons and E. G. Straus, "Dissection Graphs of Planar Point Sets", in *A Survey of Combinatorial Theory*, J. N. Srivastava *et al*, eds., North-Holland, 1973, 139-149.

[H] T. P. Hill, "Determining a fair border", *AMM* 90:7(1983)438-442.

[L] L. Lovász, "On the number of halving lines", *Ann. Univ. Sci. Budapest, Eötvos* Sect. Mat. 14, 1971, 107-108.

[M] N. Megiddo, "Applying parallel computation algorithms in the design of serial algorithms", *JACM*, 30(1983), 852-865.

[MP] D. Muller and F. Preparata, Finding the intersection of two convex polyhedra, *Theoretical Computer Science*, 7(1979), 217-236.

[OVL] M. H. Overmars and J. van Leeuwen, "Maintenance of configurations in the plane", *JCSS*, 23(1981), 166-204.

[P] F. Preparata, New Parallel Sorting Schemes, *IEEE Trans. Comput.*, C-27(1978),pp.669-673.

[V] L. Valiant, "Parallelism in Comparison Problems", *SIAM J. Comp.*, Vol.4, No.3, 1975, 348-355.

[W] D. Willard, "Polygon Retrieval", *SIAM J. Comp.* 11, 1982, 149-165.

[YB] I. M. Yaglom and V. G. Boltyanskii, *Convex Figures*, (trans.) Holt, Rinehart and Winston, 1961.

[Y] F. Yao, "A 3-Space Partition and its application", *15th STOC*, 1983, 258-263.