Tentative Prune-and-Search for Computing Fixed-Points with Applications to Geometric Computation

David Kirkpatrick Jack Snoeyink*
Department of Computer Science
University of British Columbia

Abstract

Motivated by problems in computational geometry, we investigate the complexity of finding a fixed-point of the composition of two or three continuous functions that are defined piecewise. We show that certain cases require nested binary search taking $\Theta(\log^2 n)$ time. Others can be solved in logarithmic time by using a prune-and-search technique that may make tentative discards and later revoke or certify them. This work finds application in optimal subroutines that compute approximations to convex polygons, dense packings, and Voronoi vertices for Euclidean and polygonal distance functions.

1 Introduction

Several fundamental problems in computational geometry can be expressed as a search for special k-tuple with one element drawn from each of k lists. Examples with k equal to 2 or 3 include common tangents to two convex polygons, special chords in a polygon, and Voronoi vertices (circles or polygons tangent to three given points or polygons). Many of these problems can be solved in a natural way by nested binary search. Some can be solved more efficiently by "prune-and-search." In this paper, we study general search techniques in the framework of computing fixed-points of the composition of functions, and then apply the results to geometric problems of computing chords, separation, and Voronoi vertices.

The algorithmic technique known as "prune-and-search" answers a search problem by performing some computation and pruning the amount of input data by a constant fraction. Many (perhaps most) prune-and-search algorithms, for example Megiddo's linear programming algorithm [12], perform computation on the entire input to discard a fraction. Nevertheless, computations such as binary search and Overmars and van Leeuwen's common tangent algorithm [5, 14, 15] can be viewed as prune-and-search algorithms that look at local information in $O(1)$ time to discard a fraction of the input. In section 2 we develop the *tentative prune-and-search technique* for searching k lists when local information is insufficient to determine which fraction to discard. This technique makes *tentative* decisions that are later be *certified* or *revoked.* By a potential function argument, we show that when the technique is applicable, at most $\Theta(k \log n)$ steps are required to search k lists, each containing n elements.

*Both authors supported in part by NSERC Research Grants.

The possible complexities of searching for a special k tuple in a sequence of k lists are nicely illustrated in section 3 by the problem of identifying a fixed-point of the composition of k monotone, continuous functions, each defined piecewise on its domain. In section 3.1 we give the precise form of the functions. In section 3.3 we show that these fixed-point problems can be solved in $O(\log^2 n)$ steps by nested binary search. There is a matching lower bound in certain cases where the fixed-point is not necessarily unique. When monotonicity properties imply that there is a unique fixed-point, we can use tentative prune-and-search to find it in $O(\log n)$ time for two functions (section 3.4) and three functions (section 3.5).

Many basic problems in computational geometry that seek a special k tuple can be reduced to fixed-point computations; we give several examples in section 4. In section 4.1 we sketch an algorithm that uses $\Theta(\log n)$ time to compute chords of a given length and direction in a convex n-gon, answering a question posed by Mount [13]. This improves algorithms on approximating convex polygons by rectangles [1, 17]. In section 4.2 we cast an old problem of computing the separation distance between two disjoint convex polygons [4] as a fixed-point problem.

In subsection 4.3 we solve the inscribed triangle homothet problem: given a convex polygon P, find the largest homothet of a given triangle T that can be inscribed in P. (A *homothet* of T is a scaled and translated copy of T.) This problem arises in the following manner in Kao and Mount's algorithm [8] for computing generalized Voronoi diagrams. A convex polygon P that contains the origin in its interior defines a *convex distance function* $d : E^2 \times E^2 \to \mathcal{R}$ by

$$d(p, q) = \limsup\{\alpha : \alpha(q - p) \in P\}.$$

The boundary of P is the set of unit vectors under d. To find the point equidistant from three sites under the convex distance function defined by P one wants to scale P so it passes through the three sites. If one instead fixes P and scales the triangle defined by the sites, one obtains the "largest triangle homothet" problem.

Motivated by retraction-based motion planning, Levin and Sharir [10] extended Yap's algorithm for computing the Voronoi diagram of n line segments [19] to compute Voronoi diagrams under convex distance functions. When the distance function is defined by a m-gon (for m constant), their algorithm runs in $O(n \log n)$ time and $O(n)$ space. (To be precise, it runs in $O(n \log N)$ time when the segments are organized into N connected polygons.) Linear factors of m are hidden by the big-O in both the space and time, however. Kao and Mount extend this algorithm to compute a compact representation of the Voronoi that still supports optimal point location queries. Their representation takes $O(n)$ space, with no dependence on m, and can be computed in $O(n \log n \log^2 m)$ time. Our algorithm saves a $\log m$ factor by locating the Voronoi vertices more efficiently.

In subsection 4.4, we compute Voronoi vertices: given three disjoint convex polygons, find the points that are equidistant from all three (where distance is measured by the Euclidean metric). With Michael McAllister [11], we use this as a primitive to compute an approximate Voronoi diagram of convex objects. One can obtain a piecewise-linear diagram for retraction-based motion planning whose complexity is linear in the number of objects.

Our motivating applications lead us to the study of fixed-point computations of functions defined on the unit interval under a natural but simple model of function evaluation. We do not consider here the complexity of finding fixed-points in higher dimensions or under other models of function evaluation. There is a large body of literature on such questions in mathematics and economics [2, 18].

2 General tentative prune-and-search

For our purposes, prune-and-search is a candidate elimination strategy, involving a collection of a k lists, where k is a constant. A *candidate* is a k-tuple with one element (the local candidate) drawn from each list. In the *normal mode* of operation the properties of a candidate k-tuple (including, perhaps, neighbouring elements) make possible the elimination of all elements to one side of the local candidate in one or more lists. Repeating this primitive step in a binary-searching fashion guarantees that at least one candidate list is reduced to constant size after $O(\log n)$ steps, where n denotes the maximum initial list size.

In the problems that we address in sections 3.5, 4.3 and 4.4, local properties of a candidate tuple do not suffice to eliminate a portion of any of the lists. Thus, we use a second mode, *tentative mode*, in which we have tentatively discarded portions from the ends of each list with the assurance that all the tentative discards made to at least one of the lists were correct.

We say that *normal mode is viable* under the following conditions.

- No list has a portion that has been tentatively discarded.
- We can choose middle elements from each list to form a candidate k-tuple.
- Properties of this k-tuple allow us either to discard from some list the half that is to one side of the local candidate, or to switch to tentative mode by tentatively discarding half of *each* list with the assurance that one of the discards is correct.

We say that *tentative mode is viable* if

- Every list has a portion that has been tentatively discarded.
- For any list L, we can choose a middle element of the remaining elements and, on all other lists, choose the boundary between remaining and tentatively discarded elements to form a candidate k-tuple.
- Properties of this k-tuple allow us either to permanently or tentatively discard the half of what remains of L up to the local candidate, or to certify that all of the tentative discards on one of the other lists are correct, to revoke all other tentative discards, and to return to normal mode.

We use a potential function argument to prove that, by switching between the normal and tentative modes as they become viable, we reduce some list to constant size after $O(\log n)$ steps.

Theorem 2.1 *If one of the normal and tentative modes is always viable, then tentative prune and search can reduce one of k lists of initial size n to a single element in $O(k \log n)$ steps.*

Proof: For list A, let A_T denote the number of elements tentatively discarded and A_R denote the elements remaining. Define the list potential $\Phi_A = 2\log A_R + 2(k-1)\log(A_R + A_T)$. The global potential is the sum of list potentials, plus $2k-1$ in tentative mode:

$$\Phi = \Big(\sum_{L \text{ a list}} \Phi_L \Big) + (2k-1)(\exists L_T > 0).$$

In normal mode, we either discard half of some list or else tentatively discard half of each list and enter tentative mode. The former case decreases some list potential, and therefore Φ, by $2k$. The latter decreases each list potential by 2; entering tentative mode gives a net decrease of 1.

In tentative mode, we consider the lists in round-robin order and either tentatively discard half of the current list (a permanent discard is considered tentative for the analysis) or else certify all tentative discards made to one list, revoke the rest, and return to normal mode. In the former case Φ decreases by 2. In the latter case, suppose that the certified list participated in t tentative steps, including the first. Then the potential of the certified list decreases by $2(k-1)t$. Each other list potential gains 2 for each tentative step that it participated in—since the lists were considered in round robin order, this is at most $2(t+1)$ for each of the $k-1$ lists. Since we leave tentative mode, the net decrease in Φ is at least

$$2(k-1)t + (2k-1) - 2(k-1)(t+1) = (2k-1) - 2(k-1) = 1.$$

Because the potential Φ decreases from the initial potential $\Phi = O(k \log n)$ by a constant at each step, and Φ cannot be negative, the search can proceed at most $O(k \log n)$ steps. ∎

3 Fixed-points of the composition of functions

In this section we study the complexity of finding a fixed-point of the composition of k monotone, continuous functions. We define the problem for general k, but the cases that we use are primarily $k = 2$ and $k = 3$. Depending on the character of the functions, a fixed-point either requires a nested binary search or can be found by tentative prune and search. We are grateful to Günter Rote for suggesting a functional framework.

3.1 Preliminary definitions and results

All functions in section 3 are monotone, continuous functions that are defined on ordered sets that have the topology of the closed real interval $[0, 1]$. We say that a set $\mathcal{F}$ forms a set of *basic functions* if

- The functions in $\mathcal{F}$ can be evaluated in constant time, and
- A fixed-point of the composition of k functions in $\mathcal{F}$ with appropriate domains and ranges can be computed in time that depends only on k.

The set of all linear functions is one example of a set of basic functions; usually the application determines a natural set.

Suppose that $\mathcal{F}$ is a set of basic functions. We say that a function $f\colon A \to B$ is *piecewise basic* if

- $A = [a_0, a_n]$ is partitioned into n *basic intervals* by $a_0 \le a_1 \le \cdots \le a_{n-1} \le a_n$,
- f is defined piecewise by functions $f_i\colon [a_{i-1}, a_i] \to B$ from $\mathcal{F}$.

We define the complexity of a piecewise-basic function to be the number of intervals into which its domain is partitioned: $\|f\| = n$.

When we are given a piecewise-basic function f, we assume that we are given the ordered list $a_0, a_1, \ldots, a_n$ in an array or balanced binary search tree and the individual functions $f_1, f_2, \ldots, f_n$. The ease of evaluating $f(x)$ depends on whether we know the basic interval containing x:

Observation 3.1 *If we know the basic interval $[a_{i-1}, a_i]$ containing x then we can evaluate $f(x)$ in constant time. If we know only the list of interval boundaries then to evaluate $f(x)$ takes $\Theta(\log \|f\|)$ time.*

Proof: If we know that $x \in [a_{i-1}, a_i]$, then $f(x) = f_i(x)$, and we can evaluate this basic function in constant time. We can determine an unknown basic interval by binary search. Any algorithm that does not do at least a binary search can be defeated by an adversary. ∎

In actuality, we will not need to evaluate basic functions. It will be sufficient to determine if $f_i(x)$ is less than, equal to, or greater than y for a given x in the domain and y in the codomain of f. Comparing $f(x)$ and y also takes constant time if we know the basic function that applies to x and logarithmic time if we do not. We do not stress this point in the present section, although it will become important in section 4.

We are interested in the complexity of finding fixed-points of compositions of functions that are monotone, continuous, and piecewise basic. the mean value theorem establishes the existence of a fixed-point. (Monotonicity is not required for a fixed-point to exist, but it makes our search techniques easier and all our applications use monotone functions.)

Theorem 3.2 *Let $f\colon A \to B$ be a monotone, continuous function with A and B having the topology of the closed real interval $[0, 1]$. If $A \subseteq f(A)$, then then f has a fixed-point in A.*

Proof: The function $g(x) = f(x) - x$ is continuous on A, because f is continuous, and has different signs at the endpoints of A, because f is monotone and onto A. By the mean value theorem, g has a zero, which is a fixed-point of f in A. ∎

3.2 Binary search for the fixed-point of one function

Suppose that $f\colon A \to B$ is monotone, continuous, piecewise basic, and satisfies $A \subseteq f(A)$. Then theorem 3.2 says that f has a fixed-point. We can find one by an easy binary search.

Theorem 3.3 *If $f\colon A \to B$ is a monotone, continuous, piecewise-basic function, and $A \subseteq f(A)$, then we can compute a fixed-point of f in $O(\log \|f\|)$ steps.*

Proof: Let $n = \|f\|$ and let $\{a_0, a_1, \ldots, a_n\}$ be the interval boundaries of the partition of A. We can evaluate $f(a_0)$ and $f(a_n)$ to verify that f is onto and determine if f is monotone increasing or decreasing. We maintain an interval $A' = [a_{\mathrm{lo}}, a_{\mathrm{hi}}]$ such that

$A' \subseteq f(A')$—by theorem 3.2, such an interval contains a fixed-point. Initially lo $= 0$ and hi $= n$, so $A' = A$.

Choose an index m halfway between lo and hi. We can evaluate $f(a_m) = f_m(a_m)$ in constant time. If $a_m \le f(a_m)$ and f is monotone increasing, then set lo $= m$ because f is onto $[a_m, a_{\mathrm{hi}}]$. If f is monotone decreasing, set hi $= m$ instead. The cases for $a_m > f(a_m)$ are the reverse. Thus, after $O(\log n)$ function evaluations, we have reduced $A = [a_{\mathrm{lo}}, a_{\mathrm{hi}}]$ to an interval that contains a fixed-point and is the domain for a single basic function f_{hi}. We can then compute the fixed-point of f_{hi}. ∎

3.3 Nested binary search and an $\Omega(\log^2 n)$ lower bound

If k functions $f, g, \ldots, h$ are monotone and continuous, then so is their composition $F = h \circ \cdots \circ g \circ f$. If, in addition, $f, \ldots, h$ are piecewise basic and F is onto its domain, then F has a fixed-point that we can find by nested binary search—an outer search from theorem 3.3 and an inner search to evaluate F. When F is monotone increasing and functions f through h each have complexity $\|f\| = n$, we also prove a lower bound of $\Omega(k \log^2 n)$ search steps.

Theorem 3.4 *Suppose that F, the composition of k monotone, continuous, piecewise-basic functions, is onto its domain. One can compute a fixed-point of F using $O(k^2 \log n)$ basic function evaluations and $O(k^2 \log^2 n)$ search steps.*

Proof: We maintain intervals $A_f, A_g, \ldots, A_h$ of the domains of f, g, through h that can contain a fixed-point, starting from the initial domains. We can reduce A_f by half after choosing the middle interval boundary a_m and testing if $F(a_m) < a_m$ just as in theorem 3.3.

In order to evaluate $F(a_m) = h(\cdots g(f(a_m))\cdots)$, we use $k-1$ binary searches according to observation 3.1. This reduces A_f to a single interval that contains a fixed-point after $O(k \log n)$ basic function evaluations and $O(k \log^2 n)$ search steps.

We then reduce A_g through A_h by the same technique—that is, we reduce A_g by looking for a fixed-point of $f \circ h \circ \cdots \circ g \colon A_g \to A_g$. When all domains have been reduced to single intervals then we can find a fixed-point of the composition of the basic functions defined on these intervals. ■

We can generalize a lower bound proof of Guibas et al. [6] to prove that $\Omega(k \log^2 n)$ search steps may be required for a fixed-point problem. Let k be even and let $n = m^2$ for an integer m. We describe an adversary that chooses k monotone increasing, piecewise-linear functions over $[0, 4n+1]$. Each function has complexity n.

To disclose the piecewise basic of the ith function f^i, the adversary must disclose the interval boundaries $\{a_0^i, a_1^i, \ldots, a_n^i\}$ and the basic functions $f_1^i, \ldots, f_n^i$. The adversary does so in response to queries of the form, "For function f^i, what is the jth basic function and its domain?" The response would be $(f_j^i, [a_{j-1}^i, a_j^i])$. We show that $\Omega(k \log^2 n)$ queries are required to determine which interval of one of the domains contains a fixed-point.

The adversary chooses functions that are nearly identity. Initially, the first and last basic functions for each i are defined as $f_1^i \colon [0,1] \to [-1,0]$ with $f_1^i(x) = x - 1$ and $f_n^i \colon [4n-1, 4n] \to [4n, 4n+1]$ with $f_n^i(x) = x + 1$.

For $1 \le j < m$ the adversary defines f_{mj}^i as the identity on $[4mj-1, 4mj]$ for i even, and defines $f_{mj-m/2}^i$ as the identity on $[4mj-2m-1, 4mj-2m]$ for i odd. These defined basic functions form the mortar in the "brick wall" pattern of figure 1. We say that the *brick* of an undefined basic function f_j^i is the set of undefined basic functions for f^i that are not separated from f_j^i by a defined basic function. If all functions in a brick are defined (and they will be defined to be identity functions in most cases) then we say the brick is *empty.* Finally, we say that f_j^i is the *last definable function in brick B* if f_j^i is an undefined basic function with at most one undefined function above it in B and one below in B. Since basic functions are linear and f^i is continuous, defining f_j^i will implicitly define the functions above and below and result in an empty brick.

The set of bricks in even functions (f^i with i even) with indices between $m(j-1)$ and mj, or the set of bricks in odd functions with indices between $m(j-1)-m/2$ and $mj-m/2$, is called a *row of bricks*. The adversary maintains a list $\mathcal{L}$ of rows that contain a non-empty brick. Rows appear in $\mathcal{L}$ in order of their lowest coordinate. Initially, there are $2m$ rows, alternately coming from even and odd functions.

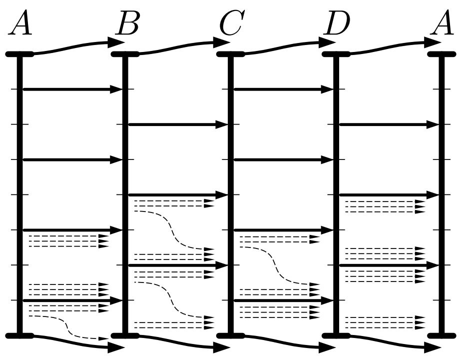

Figure 1: Defining the basic functions in bricks

Suppose that the adversary is queried for a function f^i_{mu+v}, with i even and $0 \le u, v \le m$. The adversary's strategy for answering has different cases depending on whether f^i_{mu+v} is defined, whether defining f^i_{mu+v} will empty a brick, and whether there will be a non-empty brick left in the row after defining f^i_{mu+v}. (Odd functions will be treated similarly.)

1. If f^i_{mu+v} is defined, then the adversary reports it.
2. If f^i_{mu+v} is not the last definable function in its brick B, then the adversary defines f^i_{mu+v} and at most half of the undefined functions in B. If there are fewer undefined functions below f^i_{mu+v} than above in B, then the adversary defines f^i_{mu+w} as the identity on $[4mu+w-1, 4mu+w]$ for all $1 \le w \le v$. If there are fewer above than below, then the adversary defines f^i_{mu+w} as the identity on $[4mu+3m+w-1, 4mu+3m+w]$ for all $v \le w < m$.
3. If f^i_{mu+v} is the last definable function in B, but there is another non-empty brick in the row containing B, then then the adversary defines the remaining functions in B as identity functions with appropriate domains.
4. Otherwise, defining f^i_{mu+v} will cause the B to become the last in a row of empty bricks. The adversary checks the position of this row in the list of rows $\mathcal{L}$. If the row of B is in the first half of $\mathcal{L}$, then the adversary defines all undefined functions in rows before and including the row of B so that $F(x) < x$. For example, it defines f^i_{mu+v} to map $[4mu+v-1, 4mu+3m+v]$ to $[4mu+v-1, 4mu+v]$. Then it drops these rows from $\mathcal{L}$ because they no longer contain a non-empty brick. If the row of B is in the second half of $\mathcal{L}$, then the adversary defines all functions in the row of B and after so that $F(x) > x$ and drops these rows from $\mathcal{L}$. For example, it defines f^i_{mu+v} to map $[4mu+v-1, 4mu+3m+v]$ to $[4mu+3m+v-1, 4mu+3m+v]$.

Theorem 3.5 *There exist k monotone increasing functions of complexity n, given in piecewise basic, such that $\Omega(k \log^2 n)$ search steps are required to find a fixed-point of their composition.*

Proof: If the adversary uses the strategy described above, then the algorithm cannot know the row containing the fixed-point until the adversary's list $\mathcal{L}$ has been reduced to a single row. Reducing $\mathcal{L}$ requries $\Omega(k \log^2 n)$ queries as follows: to halve the size of $\mathcal{L}$ requires emptying k bricks on a row, and to empty a brick requires $\Omega(\log \sqrt{n}) = \Omega(\log n)$ queries. Since each brick is independent, these bounds multiply. ∎

Note that this lower bound argument can be applied to a mixture of monotone increasing functions with an even number of monotone decreasing functions. When the number of monotone decreasing functions is odd, the composition is monotone decreasing. For such compositions with two or three functions we can beat the lower bound.

3.4 Prune-and-search for two functions

We can find a fixed-point of a composition of a decreasing and an increasing function in piecewise basic by a prune-and-search algorithm—with a constant time test we can eliminate half of the intervals of the domain of one of the functions.

Theorem 3.6 *Suppose that we are given two functions in piecewise basic: f monotone decreasing over the domain A and g monotone increasing over the domain B. If $B \subseteq f(A)$ and $A \subseteq g(B)$, then we can determine a fixed-point of $g \circ f$ in $O(\log \|f\| + \log \|g\|)$ steps.*

Proof: The compositions $g \circ f$ and $f \circ g$ are onto their domains and thus have fixed-points by theorem 3.2. For simplicity of notation, we assume that $n = \|f\| = \|g\|$ and let $\{a_0, a_1, \ldots, a_n\}$ and $\{b_0, b_1, \ldots, b_n\}$ be the partitions of the initial domains A and B, respectively. We maintain intervals $A' \subseteq A$ and $B' \subseteq B$ that must contain a fixed-point. Initially, $A' = [a_0, a_n]$ and $B' = [b_0, b_n]$.

Choose middle boundaries $a_m \in A'$ and $b_m \in B'$ and, in constant time, determine the relationships between $f(a_m) = f_m(a_m)$ and b_m, and between $g(b_m)$ and a_m.

We can think of the map $g \circ f$ as a twisted ribbon from A to B followed by a straight ribbon back onto A. We can flip B and cycle A and B to obtain a configuration with $f(a_m) > b_m$ and $g(b_m) > a_m$ as depicted in figure 2. (In formulæ, this transformation defines $g'(x) = b_n + b_0 - f(x)$ and $f'(x) = g(b_n + b_0 - x)$.) In this configuration, we can discard the boundaries in A' that are less than a_m—for any $a \leq a_m$, we have $f(a) \geq b_m$ and thus $g(f(a)) \geq g(b_m) > a_m$. Therefore, we are not losing fixed-points by discarding intervals before a_m.

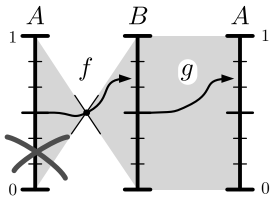

Figure 2: Discarding half of A'

We repeat this test until A' or B' is reduced to a basic interval. The corresponding basic function can be evaluated in constant time whenever necessary, so the remaining domain can be reduced to a basic interval by binary search according to theorem 3.3. Finally, a fixed-point of the composition of two basic functions can be obtained in constant time. ∎

3.5 Tentative prune-and-search for three functions

With three functions, local information may be insufficient to identify an interval that can be discarded. We turn to the tentative prune-and-search technique, described in section 2.

Suppose that we have f, g, and h, which are three monotone decreasing functions in piecewise basic and onto their respective domains. Then the composition $h \circ g \circ f$ is monotone decreasing and onto its domain; it has a fixed-point by theorem 3.2. We maintain intervals A, B, and C of the domains of f, g, and h that may contain a fixed point.

We use tentative prune-and-search to reduce one of intervals to a basic interval, after which we can apply theorem 3.6. Lemmas 3.7 and 3.8 show that the normal and tentative modes are viable by describing the computations for these modes.

Lemma 3.7 *If there are no tentative discards, then normal mode is viable.*

Proof: We must choose midpoints and either discard half of some domain or tentatively discard half of every domain while guaranteeing that one of the tentative discards is correct. Let A, B, and C be intervals of the domains of f, g, and h, respectively, that contain a fixed-point. The initial intervals are the domains: $A = [a_0, a_n]$, $B = [b_0, b_n]$, and $C = [c_0, c_n]$. (We assume that each function has complexity n.)

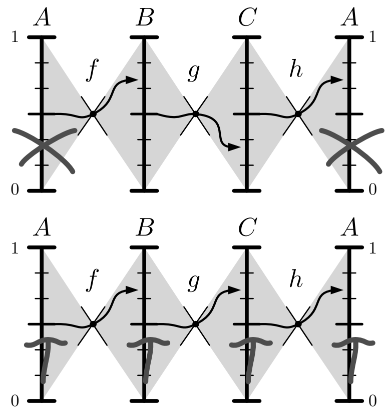

Figure 3: Discards for inconsistent and consistent comparisons

Choose middle points $a \in A$, $b \in B$ and $c \in C$ and evaluate the corresponding basic functions to determine the relationships between $f(a)$ and b, between $g(b)$ and c, and between $h(c)$ and a. As depicted schematically in figure 3, these comparisons can be consistent (all greater or less than) or inconsistent (two one way and one the other). In the inconsistent case, we can assume, without loss of generality, that $f(a) > b$, $g(b) < c$, and $h(c) > a$ as in figure 3. But then, for any $a' \in A$ less than a, $f(a') > f(a) > b$, so $g(f(a')) < g(b) < c$, and therefore $h(g(f(a'))) > h(c) > a > a'$. In words, there is no fixed-point less than a, and half of A can be discarded. This would be a discard in normal mode.

In the consistent case, suppose without loss of generality that all relationships are "greater than." Then the fixed-point configuration can have at most one member less than the middle points: If $b' < b$ then $g(b') > g(b) > c$ and $f^{-1}(b') > f^{-1}(b) > a$, and similar arguments hold if $a' < a$ or $c' < c$. We can therefore tentatively discard the portions of A, B, and C below the middle points and be assured that we are making a mistake on at most one domain. Once we do so, we enter tentative mode. ∎

In tentative mode, we refine one of the domains—introducing middle point $\hat{b} \in B$, for example. We then have four cases to consider, since $\hat{b}$ could be introduced either above or below $f(a)$, and $g(\hat{b})$ could be above or below c.

Lemma 3.8 *After leaving normal mode, tentative mode is viable.*

Proof: In tentative mode, we either discard or tentatively discard half of the domain under consideration or we certify all tentative discards made to one of the domains and return to normal mode.

Assume that we have entered the tentative mode with comparisons $f(a) > b$, $g(b) > c$, and $h(c) > a$, as illustrated in figure 3. Suppose that we refine the remaining part of B by locating the middle point $\hat{b}$. We have four cases to consider for the location of $\hat{b}$.

Case 1: $f(a) > \hat{b}$ and $g(\hat{b}) > c$. All comparisons are consistent, with $\hat{b}$ playing the role of b. Therefore, we extend the tentative discard to $\hat{b}$, assured by the argument of the lemma 3.7

that we have made a mistake on at most one domain.

Case 2: $f(a) > \hat{b}$ and $g(\hat{b}) < c$. The comparisons are inconsistent in the same way as in figure 3. Using the argument of lemma 3.7, we know that the portion of A that is less than a does not contain a fixed-point. We can then certify the tentative discard made to that portion of A, revoke all other tentative discards, and return to normal mode.

Case 3: $f(a) < \hat{b}$ and $g(\hat{b}) > c$. The comparisons are again inconsistent with two '>' and one '<'. In this case, the tentative discard to the portion of C less than c is certified and we return to normal mode by revoking any other tentative discards.

Case 4: $f(a) < \hat{b}$ and $g(\hat{b}) < c$. The comparisons are inconsistent with two '<' and one '>'. In this case the portion of B that is greater than $\hat{b}$ can be (permanently) discarded.

■

From these lemmas and theorems 2.1 and 3.6 we derive theorem 3.9.

Theorem 3.9 *Given piecewise-basic, monotone-decreasing, continuous functions f, g, and h that are onto their respective domains, we can find a fixed-point of $h \circ g \circ f$ in $O(\log \|f\| + \log \|g\| + \log \|h\|)$ steps.*

Proof: Let n denote the maximum complexity of f, g, and h. Lemmas 3.7 and 3.8 show the viability required by theorem 2.1, so in $O(\log n)$ steps we can reduce one domain to a basic interval. We can then replace one function by a basic function that can be evaluated in constant time. This reduces the problem to a fixed-point of two functions, which can be solved in $O(\log n)$ additional steps. ■

We remark that tentative prune-and-search can also be used to obtain logarithmic-time algorithms that do not fit into this functional framework. Examples include computing the common tangents of separated polygons without using a specific separating line, computing the shortest segment that joins two convex polygons while avoiding two others [9], and computing a local minimum or maximum for convex polygon width [16].

4 Geometric applications

In this section we give geometric applications of our fixed-point algorithms. We parameterize convex, plane polygons and define functions so that the configuration we seek (chords, parallel tangents, inscribed triangles, circular tangents) are fixed-points of the composition of our functions. Recall that our fixed-point algorithms do not require that these functions be able to be evaluated—we are free to pick any function, as long as the image of a given point in the domain can be compared to a given point in the range.

4.1 Computing longest and specified chords

Given a convex n-gon P, Mount [13], in his algorithm for computing double-lattice packings, sought chords parallel to direction α that were half the length of the longest chord. We describe a logarithmic-time algorithms for the longest chord parallel to α and parallel chords with specified lengths. These algorithms improve the running time of algorithms that approximate convex polygons by rectangles [1, 17].

Let us assume that α is vertical and polygon P is given by a hierarchical representation [3], which here means that "middle points" of a polygonal chain and their tangents are available. Hierarchical representations can be implemented by storing the vertices of P in order in an array or storing vertices and tangents in a balanced binary search tree.

We characterize a longest chord in P and then show how to find one.

Lemma 4.1 *The tangents to the endpoints of a longest vertical chord in a convex polygon P can be chosen to be parallel.*

Theorem 4.2 *Given an hierarchical representation of a convex n-gon P, one can find the longest chord parallel to a query segment in $\Theta(\log n)$ steps.*

Proof: Assume that the query is vertical. Break P (conceptually, at least) into above and below monotone chains, A and B, by using binary search to find the leftmost and rightmost vertices of P. Parameterize A and B by x-coordinate.

We now define functions $g : B \to \mathcal{R}$ and $f : A \to \mathcal{R}$ piecewise on the edges of A and B so that g is monotone increasing, f is monotone decreasing, $\|f\| + \|g\| = n$, and a fixed-point of $g \circ f$ satisfies lemma 4.1. The function g is simply the identity; for a point in B with parameter b, the value $g(b) = b$ is the parameter of the point $a \in A$ on the same vertical line as b as in figure 4. For a point $a \in A$, the value $f(a)$ is the x-coordinate of a point $b \in B$ such that a and b have parallel tangents. (This is not well-defined when a is a vertex and not continuous when b is a vertex. We can avoid these problems by making A and B smooth spline curves and taking the limit as these splines approach the polygon. An alternative is to use the kinetic framework [7] to parameterize point/tangent pairs as we will do in the next section.) Clearly, a fixed-point $a \in A$ such that $g(f(a)) = a$ gives points of A and B with the same x-coordinate and tangent slope.

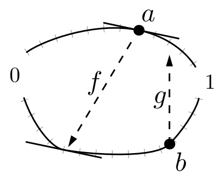

Figure 4: Defining f and g

It is easy to compare functions f and g at given points $a \in A$ and $b \in B$ without evaluating them. To compare $f(a)$ and b we compare slopes of tangents at a and b; to compare a and $g(b)$ we compare x-coordinates of a and b. Therefore, we can use theorem 3.6 to compute a fixed-point in logarithmic time. ∎

A polygonal version of the discard step of theorem 3.6 is illustrated in figure 5. Choose $a \in A$ and $b \in B$ and let p be the intersection of the tangents at a and b. If the x-coordinate order is $a.x < b.x < p.x$, then discard the portion of B to the right of b.

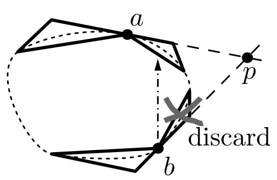

Figure 5: Discard step

Next, we show how to compute a chord with a specified direction and length in logarithmic time.

Theorem 4.3 *Given an hierarchical representation of a convex n-gon P, one can find the at most two positions where translates of a segment T are chords of P in $\Theta(\log n)$ time.*

Proof: Again, assume that the direction of T is vertical. Apply theorem 4.2 to make sure that the longest chord parallel to T is longer than T, then cut the polygon P in two along the longest chord and look for one translate of T in each piece.

Consider the piece with the longest chord on the left and split it into above and below chains A and B by finding the rightmost extreme point. If we would translate the chain A downward by the length of T, then A and B intersect at the base of the specified chord.

From lemma 4.1, we know that a pair of parallel tangents to P touch the ends of the longest vertical chord of P. The slope of these tangents separates the slopes of the chains A and B. By a skew transformation, we can make the slopes of A negative and of B positive without changing the x-coordinate of their intersection. The result is shown in figure 6.

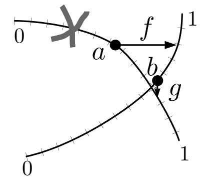

Figure 6: Chains A and B

Now, we are looking for the intersection of two chains that are both x and y monotone. We parameterize A and B by x-coordinate and define $g(b)$ as the identity and $f(a)$ to be the x coordinate of the point $b \in B$ with the same y coordinate as $a \in A$. Thus, a fixed-point of $g \circ f$ gives points of A and B with the same x and y coordinates. Notice that f is monotone decreasing, g is monotone increasing, and, if edges determine the basic functions, $\|f\| + \|g\| = n$. Because we can compare function values with points by comparing the appropriate coordinates, we can find a fixed-point in logarithmic time using the algorithm of theorem 3.6. ∎

4.2 Computing separation of convex polygons

Edelsbrunner [4] gave a logarithmic-time algorithm for computing the Euclidean closest pair of points on two disjoint, convex, n-sided polygons in the plane. Dobkin and Kirkpatrick [3] gave a unified framework using hierarchical representations. We show how to use fixed-points and also how to extend the computation to L_p norms and convex distance functions.

We begin with some points that can help in parameterization.

Lemma 4.4 *Given two disjoint, convex n-gons, A and B, in logarithmic time we can compute points $a' \in A$ and $b' \in B$ that lie on the convex hull of $A \cup B$.*

Proof: Choose arbitrary points $a \in A$ and $b \in B$ and compute the intersection of the line $\overline{ab}$ with A and B by binary search. The extreme points of these intersections give the desired a' and b'. ∎

We look at the problem of finding the smallest homothet of a closed convex set M that touches A and B. We can characterize the smallest homothet by separating tangents.

Lemma 4.5 *The smallest homothet of a closed convex set M that touches polygons A and B has parallel tangents that separate A from M and B from M.*

Theorem 4.6 *Suppose that we have two disjoint, convex, n-sided polygons A and B, and a convex set M, for which it takes T_M time to compute the two tangents of a given slope. We can compute the smallest homothet of M that touches A and B in $O(T_M \log n)$ time.*

Proof: Break A and B into polygonal chains at the points a' and b' given by lemma 4.4 and parameterize them counterclockwise over, say, reals in $[0, 1]$. We can use the kinetic framework proposed by Guibas, Ramshaw and Stolfi [7] to parameterize point/tangent pairs as we traverse a chain so that we can interchangeably use parameter values of points and tangents without worrying about the fact that polygon vertices have many tangents and tangents to edges have many points of tangency.

We further reduce A by computing the convex hull of $A \cup \{b'\}$—using binary search for the tangents to A through b'—and keeping only the portion of A inside the hull. Similarly, we keep only the portion of B inside the hull of $B \cup \{a'\}$. (This step could be avoided by complicating the definitions the functions f and g in the next paragraph.) The points that we discard do not have tangents that separate A and B, so lemma 4.5 says that they cannot be the contacts with the smallest homothet of M. We are left with chains that have at most one tangent for each slope.

We now define two functions $f\colon A \to R$ and $g\colon B \to \mathcal{R}$ with $\|f\| + \|g\| = n$ that are depicted in figure 7. For $b \in B$, let $g(b)$ be the parameter of a point a such that a and b have parallel tangents. Because we have parameterized counterclockwise, g is monotone increasing. For $a \in A$ we define $f(a)$ as follows. Let τ_a denote the tangent at a. Find the two tangents to M that are parallel to τ_a and let μ denote the ray that the points of tangency on M determine such that if the tail of μ is at a then tangent τ_a separates μ and A. Then shoot from a in the direction μ and let $f(a)$ be the parameter of the point of B encountered. If the ray misses B, then assign any value to $f(a)$ that keeps f monotone decreasing.

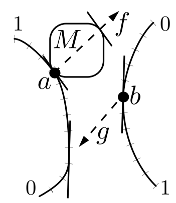

Figure 7: Functions for separation

By comparing the slopes of tangents at $a \in A$ and $b \in B$, we can compare a with $g(b)$ in constant time. By finding the tangents to M parallel to $a \in A$ and comparing the projections of a and b in the direction of the ray μ, we can compare $f(a)$ with b in T_M time. Thus, theorem 3.6 gives us a way to compute a fixed-point of $g \circ f$ in $O(T_M \log n)$ time. This fixed-point satisfies the characterization lemma 4.5 and gives the contact points for the smallest homothet of M that touches A and B. ∎

Using the unit ball for the L_p norm, we obtain the shortest distance between two polygons.

Corollary 4.7 *One can compute the L_p shortest distance between two disjoint, convex n-gons in $O(\log n)$ time.*

4.3 Inscribed triangle homothets

We can find a fixed-point of a composition of three functions to give a logarithmic-time algorithm for the *inscribed triangle homothet problem:* Given a convex polygon P, compute points a', b' and c' on the boundary $\partial(P)$ such that $\triangle a'b'c'$ is homothetic to a given triangle $\triangle abc$.

We begin by computing the homothet of $\triangle abc$ that circumscribes P: we compute the tangents parallel to the edges of $\triangle abc$ in logarithmic time by binary search. We use the points of tangency to delimit three chains, A, B and C, where the vertices a', b' and c' must lie. We can obtain the hierarchies for these three chains from the hierarchy for P. The chain A is monotone with respect the directions $\overrightarrow{a'b'}$ and $\overrightarrow{a'c'}$; analogous statements hold for B and C.

Theorem 4.8 *One can compute the homothet of $\triangle abc$ that is inscribed in an n-gon P in $\Theta(\log n)$ time.*

Proof: Parameterize the chains A, B, and C counterclockwise over $[0,1]$. The function value $f(a')$ is the parameter of the point where the ray from a in direction $\overrightarrow{ab}$ intersects the chain B. Given $b' \in B$, we compare $f(a')$ and b' in constant time by projecting a' and b' parallel to $\overrightarrow{ab}$. The function $f\colon A \to B$ is well-defined because B is monotone with respect to the direction $\overrightarrow{ab}$. We define g and h by shots along $\overrightarrow{bc}$ and $\overrightarrow{ca}$, respectively. Each of these functions is continuous and decreases monotonically.

We can find the fixed-point of a composition of basic functions in constant time: Given three segments on which a', b', and c' can lie, we represent each segment as a linear combination of its endpoints. For example, segment A with endpoints a_0 and a_1 is $A(s) = sa_0 + (1-s)a_1$. There is a homothet $\triangle a'b'c'$ if the three simultaneous linear equations

$$\frac{A(s)-B(t)}{a-b} = \frac{B(t)-C(u)}{b-c} = \frac{C(u)-A(s)}{c-a}$$

have a solution with $0 \le s,t,u \le 1$.

Therefore, theorem 3.9 says that we can find a fixed-point in logarithmic time. ∎

The case that requires tentative prune-and-search is shown in figure 8. Local information is not sufficient to discard half of some chain, but by discarding the clockwise portions of each chain we can be assured of making at most one mistake.

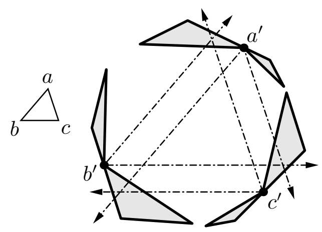

Figure 8: The case that needs tentative discards

Theorem 4.8 computes the largest homothet of T that is inscribed in P: all three vertices are on $\partial(P)$. The largest homothet of T that is contained in P may be larger if one of its edges is a longest chord in P. These cases can be checked using the algorithm of theorem 4.2.

Corollary 4.9 *One can compute the largest homothet of a triangle T in P using $\Theta(\log n)$ time.*

4.4 Computing Voronoi vertices

In this section, we give a logarithmic-time algorithm for the Voronoi vertex problem: given disjoint convex polygons P, Q and R, determine the points in the plane that are equidistant from P, Q and R. We assume that we are given representations of P, Q and R that allow access to middle points and normals and that their total description has size $O(n)$.

Theorem 4.10 *The Voronoi vertex problem in the plane can be solved in $O(\log n)$ time by tentative prune-and-search.*

Proof: As the problem definition suggests, there may be more than one point (or none) satisfying the Voronoi vertex property. Straightforward preprocessing, involving the construction of the closest pairs on the boundaries of each pair of polygons as well as pairwise outer common tangents, allows us to restrict our attention to three convex chains, A, B and C, from P, Q and R respectively, for which the Voronoi vertex problem has a unique solution.

Parameterize the chains A, B, and C clockwise over $[0,1]$. (Counterclockwise, if viewed from the outside of P, Q and R. The function value $f(a')$ is defined to be the parameter of the point $b' \in B$ whose normal intersects the normal to A at a' in a point q such that the lengths $a'q = b'q$. (If a point has more than one normal, we choose one in a canonical fashion; e.g., furthest ccw.) Equivalently, the normal at a' intersects the bisector of A and B at some point q; the function value $f(a')$ is the parameter of the point of B closest to q. Functions g and h are defined analogously.

We certainly do not know how to evaluate such functions efficiently, but given a' and b' we can determine whether $f(a') > b'$: Simply compute the point p that is the intersection of the normals at a' and at b' and determine whether $a'p < b'p$. If so, then the normal at a' intersects the bisector beyond p and $f(a') > b'$ on B. This case is illustrated in figure 9.

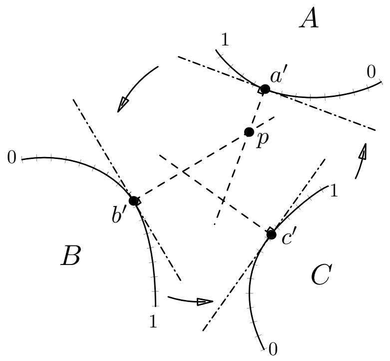

Figure 9: Normals at a, b, and c

Once again, we have defined continuous, monotone-decreasing functions. The fixed-point of a composition of basic functions—the disk tangent to A, B, and C—can be determined by algebra when when edges containing $a' \in A$, $b' \in B$ and $c' \in C$ are known. Thus, by theorem 3.9, we can compute a Voronoi vertex in $O(\log n)$ time. ∎

5 Conclusions

In this paper we have studied the complexity of computing fixed-points of compositions of functions that are defined piecewise. This functional framework captures the complexity of several basic problems in computational geometry, such as computing chords, separation, inscribe triangles, and Voronoi vertices for disjoint convex polygons in the plane. From this study also comes a general multiple-list search technique that we call tentative prune-and-search.

We are still exploring the generalizations and extensions of the algorithms to compute fixed-points. Although all our applications used monotone functions, the correctness proofs for some (and perhaps all) of the searches could be generalized to continuous (non-monotone) piecewise basic functions. The most interesting extension would be to, in logarithmic time, compute a fixed-point of the composition of more than three monotone, continuous functions when that composition is monotone decreasing.

Acknowledgements

We thank David Mount for introducing us to the specified chord and inscribed homothet problems and thank Günter Rote for suggesting a the functional framework for tentative discards involving three lists. Thorsten Graf suggested the problem of separation under the L_p norm.

References

[1] H. Alt, J. Blömer, M. Godau, and H. Wagener. Approximation of convex polygons. In *Seventeenth ICALP*, number 443 in LNCS, pages 703–716. Springer-Verlag, 1990.

[2] K. C. Border. *Fixed point theorems with applications to economics and game theory.* Cambridge University Press, 1985.

[3] D. P. Dobkin and D. G. Kirkpatrick. Determining the separation of preprocessed polyhedra: A unified approach. In *Seventeenth ICALP*, number 443 in LNCS, pages 400–413. Springer-Verlag, 1990.

[4] H. Edelsbrunner. Computing the extreme distances between two convex polygons. *J. Alg.*, 6:213–224, 1985.

[5] H. Edelsbrunner and H. Maurer. Finding extreme points in three dimensions and solving the post-office problem in the plane. *Info. Proc. Let.*, pages 39–47, 1985.

[6] L. Guibas, J. Hershberger, and J. Snoeyink. Compact interval trees: A data structure for convex hulls. *Int. J. Comp. Geom. App.*, 1(1):1–22, 1991.

[7] L. Guibas, L. Ramshaw, and J. Stolfi. A kinetic framework for computational geometry. In *Proc. 24th FOCS*, pages 100–111, 1983.

[8] T. C. Kao and D. M. Mount. An algorithm for computing compacted Voronoi diagrams defined by convex distance functions. In *Proc. Third Can. Conf. Comp. Geom.*, pages 104–109, Simon Fraser University, Vancouver, 1991.

[9] D. Kirkpatrick and J. Snoeyink. Computing constrained segments: Butterfly wingspans in logarithmic time. In *Proc. Fifth Can. Conf. Comp. Geom.*, pages 163–168, Waterloo, Canada, 1993.

[10] D. Leven and M. Sharir. Planning a purely translational motion for a convex polygonal object in two dimensional space using generalized Voronoi diagrams. *Disc. & Comp. Geom.*, 2:9–31, 1987.

[11] M. McAllister, D. Kirkpatrick, and J. Snoeyink. A compact piecewise-linear Voronoi diagram for convex sites in the plane. Accepted to the FOCS, 1993.

[12] N. Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM J. Comp.*, 12:759–776, 1983.

[13] D. M. Mount. The densest double-lattice packing of a convex polygon. In J. E. Goodman, R. Pollack, and W. Steiger, editors, *Discrete and Computational Geometry: Papers from the DIMACS Special Year*, pages 245–262. American Mathematical Society, Providence, RI, 1991.

[14] M. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comp. Sys. Sci.*, 23:166–204, 1981.

[15] F. P. Preparata and M. I. Shamos. *Computational Geometry—An Introduction*. Springer-Verlag, New York, 1985.

[16] G. Rote, C. Schwarz, and J. Snoeyink. Maintaining the approximate width of a set of points in the plane. In *Proc. Fifth Can. Conf. Comp. Geom.*, pages 258–263, Waterloo, Canada, 1993.

[17] O. Schwarzkopf, U. Fuchs, G. Rote, and E. Welzl. Approximation of convex figures by pairs of rectangles. In *Proc. 7th Sympos. Theoret. Aspects Comput. Sci.*, volume 415 of *Lecture Notes in Computer Science*, pages 240–249. Springer-Verlag, 1990.

[18] S. A. Vavasis. *Complexity of fixed point computations*. PhD thesis, Stanford University, 1989.

[19] C. K. Yap. An $O(n \log n)$ algorithm for the Voronoi diagram of a set of simple curve segments. *Disc. & Comp. Geom.*, 2:365–393, 1987.