Synergistic Solutions for Merging and Computing Planar Convex Hulls

Jérémy Barbay(✉) and Carlos Ochoa

Departamento de Ciencias de la Computación, Universidad de Chile, Santiago, Chile
jeremy@barbay.cl, cochoa@dcc.uchile.cl

Abstract. We describe and analyze the first adaptive algorithm for merging k convex hulls in the plane. This merging algorithm in turn yields a synergistic algorithm to compute the convex hull of a set of planar points, taking advantage both of the positions of the points and their order in the input. This synergistic algorithm asymptotically outperforms all previous solutions for computing the convex hull in the plane.

Keywords: Convex hull · Merging · Multivariate analysis · Synergistic

1 Introduction

One way to close the gap between practical performance and the worst case complexity over instances of fixed input size is to refine the latter, considering smaller classes of instances defined via difficulty measures. The computation of the Convex Hull of a set of n points in the plane is a good example of the variety of such techniques. The `Gift Wrapping` algorithm proposed by Chand and Kapur [6] in 1970 is adaptive to the size h of the Convex Hull output, with a running time within $O(nh) \subseteq O(n^2)$. In 1973, Graham [8] described an algorithm known as `Graham's scan`, running in time within $O(n \log n)$. On instances where the output is small ($h \in o(\log n)$), `Gift Wrapping` asymptotically outperforms `Graham's scan`, while the reverse is true on other instances.

In 1986 (13 years later!), Kirkpatrick and Seidel [9] described an algorithm computing the Convex Hull of size h in time within $O(n \log h)$, which asymptotically outperforms both `Gift Wrapping` and `Graham's scan`. This was further improved when Afshani et al. [1] observed that a minor variant of Kirkpatrick and Seidel's algorithm [9] takes optimal advantage of the positions of the points, and proved its instance optimality among algorithms ignoring the order of the input, in a decision tree model where the tests involve only multilinear functions with a constant number of arguments. They showed that the time complexity of this variant is within $O(n(1 + \mathcal{H}(n_1, \dots, n_h))) \subseteq O(n(1 + \log h))$, where $n_1, \dots, n_h$ are the sizes of a partition of the points by enclosing triangles,

C. Ochoa is supported by CONICYT-PCHA/Doctorado Nacional/2013-63130161 (Chile).

such that every triangle is completely below the upper hull of the points, with the minimum possible value for $\mathcal{H}(n_1, \ldots, n_h) = \sum_{i=1}^{h} \frac{n_i}{n} \log \frac{n}{n_i} \leq \log h$.

Levcopoulos et al. [10] described in 2002 an algorithm to compute the convex hull of a set of planar points that do consider the order of the input, and thus could break Afshani et al.'s lower bound [1]. The algorithm uses a decomposition of the points into simple polygonal chains. A *polygonal chain* is specified by a sequence of points, and consists of the line segments connecting the pairs of consecutive points. A polygonal chain is *simple* if it does not have a self-intersection. They prove that the time complexity of this algorithm is within $O(n(1+\log \kappa))$, where κ is the minimum number of simple subchains into which the sequence of n points can be partitioned. Note that κ depends only of the input order: by reordering the points, one can always reduce it to one, or increase it to within $\Theta(n)$.

A dovetailing combination[1] of the algorithms described by Kirkpatrick and Seidel [9] and Levcopoulos et al. [10] takes advantage of the order in which the points are given while maintaining (input order oblivious) instance optimality. But this solution is inefficient: many operations will be repeated, and some opportunities to quickly solve the instance are lost due to this lack of communication between the two parallel branches of the algorithm. To address this problem, we describe an algorithm for computing the Convex Hull of a set of planar points that takes advantage both of the positions of the points and their order in the input, synergistically, in the sense that it never performs asymptotically worse than the algorithms described by Kirkpatrick and Seidel [9] and Levcopoulos et al. [10], and on large classes of instances asymptotically outperforms both by more than a constant factor (see Example 1 in Sect. 3.2).

In order to yield a synergistic algorithm, we obtain several new results. (1) We generalize Demaine et al.'s algorithm and corresponding analysis [7] from the Merging of Multisets to the Merging of Convex Hulls, in Sect. 2 (this is the most technical part of this work). (2) We present an algorithm to partition a sequence of points into simple subchains, which is faster than the one described by Levcopoulos et al. [10], and (3) we refine Levcopoulos et al.'s measure of difficulty and analysis, in Sect. 3.1. (4) We combine those results into a synergistic algorithm to compute the Convex Hull in the plane, and (5) we prove that in large classes of instances this algorithm asymptotically outperforms the best previous solutions [1,10], and never asymptotically performs worse than them, in Sect. 3.2.

2 Computing the Union of Upper Hulls

The computation of convex hulls in the plane reduces to the computation of upper hulls [9]. Given ρ upper hulls in the plane, where the points in each upper hull are given sorted by their x-coordinates, the Union of Upper Hull problem consists in computing the upper hull of the union of the ρ upper hulls.

[1] A dovetailing combination of k algorithms executes the k algorithms in parallel and stops as soon as one of the algorithms finishes.

Algorithm 1. `Quick Union Hull`

```
Input: A set $\mathcal{U}_1, \ldots, \mathcal{U}_\rho$ of $\rho$ upper hulls
Output: The upper hull of the union of $\mathcal{U}_1, \ldots, \mathcal{U}_\rho$
1: Compute the median $\mu$ of the slopes of the middle edges of the $\rho$ upper hulls;
2: Identify the "pivot" point $p$ in the input that has a supporting line of slope $\mu$;
3: Partition the $\rho$ upper hulls by the vertical line through $p$;
4: For each upper hull $\mathcal{V}$, compute the (at most) two tangents of $\mathcal{V}$ through $p$: the
   ones to the left and right of $p$, and discard the blocks of consecutive points below
   the line segments determined by the points of tangency;
5: Output a block of points in the upper hull $\mathcal{U}$ containing $p$ that forms part of the
   union, by computing common tangents between $\mathcal{U}$ and the other upper hulls;
6: Discard all points that lie below the line segments determined by the points in the
   common tangents between $\mathcal{U}$ and the other upper hulls;
7: Recurse on the resulting upper hulls to the left and to the right of $p$.
```

We describe the algorithm `Quick Union Hull`, that solves the Union of Upper Hull problem, in Sect. 2.1, and we analyze its time complexity in Sect. 2.2. This algorithm is inspired by the algorithms `Simplified Ultimate Planar Convex Hull` described by Chan et al. [5], and `Quick Synergy Sort` described by Barbay et al. [3]. The `Quick Union Hull` algorithm is an essential building block towards the synergistic algorithm for computing the convex hull of a set of planar points, described and analyzed in Sect. 3.

2.1 Description of the Algorithm Quick Union Hull

In the context of the Union of Upper Hull problem, in each upper hull the points are given sorted by their x-coordinates, and the slopes of the edges monotonically decrease from left to right. The algorithm `Quick Union Hull` takes advantage of these facts: its pseudocode is described in Algorithm 1. For each upper hull, the algorithm identifies blocks of consecutive points that form part of the output and blocks of consecutive points that lie underneath the upper hull of the union. The algorithm uses a divide-and-conquer approach to take advantage of the positions of the points.

We next define some key concepts that are used in the description of the algorithm. Let S be a finite set of planar points. A *supporting line* of S is a straight line that contains a point p of S and that leaves all the points of S in the same half-plane (i.e., p is a vertex of the convex hull of S). Let μ be the slope of a supporting line passing through a point q in the upper hull $\mathcal{U}$ of S. If there is a pair of points of S to the left of q such that the line through the pair has slope less than μ, then the rightmost point in the pair cannot be part of $\mathcal{U}$. A symmetric situation arises if the pair of points is to the right of q and the slope of the line through the pair is greater than μ: the leftmost point in the pair cannot be part of $\mathcal{U}$. If the points in S are paired, a good candidate to discard points that cannot be part of $\mathcal{U}$ is the point p that has a supporting line whose slope is the median of the slopes of the lines passing through the pairs.

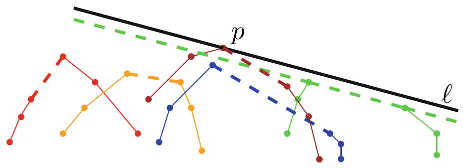

Fig. 1. An instance of the Union of Upper Hulls problem. The middle edges of the upper hulls are marked by thick dashed segments, and the one whose slope is the median μ of the slopes of the middle edges has been extended into a line. The straight line ℓ is the supporting line of slope μ. The line ℓ passes through the "pivot" vertex p.

Once the points have been discarded, the choice of p guarantees that at most a constant fraction of the points in S remains on each side of p [5].

In the Union of Upper Hulls problem, in each upper hull, the slopes of the edges monotonically decrease from left to right. So, in each upper hull $\mathcal{V}$, the edge at the middle position is the one whose slope is the median among the slopes of the edges of $\mathcal{V}$. We show that the point that has a supporting line whose slope is the median of the slopes of the middle edges of the upper hulls is also a good candidate to discard points that cannot be part of the upper hull of the union. Note that the time complexity of computing the median of the slopes of the middle edges of the upper hulls is linear in the number ρ of upper hulls, but that the time complexity of pairing the points and computing the median of the slopes of the lines through the pairs is linear in the number n of points [5].

The algorithm `Quick Union Hull` identifies a "pivot" vertex p of the upper hull of the union, and uses p to discard blocks of consecutive points that cannot be part of the output. It computes the median μ of the slopes of the middle edges of the upper hulls, and identifies p as the point that has a supporting line ℓ of slope μ. Note that p is the extreme point in the direction orthogonal to ℓ. Taking advantage that in each upper hull $\mathcal{V}$ the slopes of the edges are sorted, the algorithm identifies the extreme point in the direction orthogonal to ℓ by performing a doubling search[2] for the value μ in the list of slopes of the edges of $\mathcal{V}$. (See Fig. 1 for a graphical representation of these steps.)

To know which points are to the left and which ones are to the right of p, the algorithm partitions the points in the upper hulls by the vertical line $x = p_x$, where p_x is the x-coordinate of the point p, by performing doubling searches for the value p_x in the x-coordinates of the points in the upper hulls.

For each upper hull $\mathcal{V}$, the algorithm then computes the (at most) two tangents of $\mathcal{V}$ through p: the one passing through a point to the left of p in $\mathcal{V}$, and the one passing through a point to the right of p in $\mathcal{V}$. In $\mathcal{V}$, the algorithm discards the blocks of consecutive points below the line segments determined by the points of tangency. It computes all the tangents via doubling searches [2].

Before the recursive step, in the upper hull $\mathcal{U}$ containing p, the algorithm identifies a block $\mathcal{B}$ of consecutive points that forms part of the output (p is

[2] Doubling search is a technique for searching sorted unbounded arrays in which an element of rank k is found by performing $2 \log k$ comparisons [4].

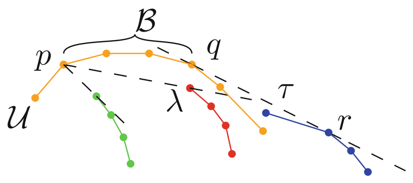

Fig. 2. The state of the algorithm `Quick Union Hull` during an execution of the step that computes the block $\mathcal{B}$ that forms part of the upper hull of the union. The upper hull $\mathcal{U}$ contains the point p. λ marks the tangent of maximum slope between p and the upper hulls to the right of p. τ marks the common tangent between the portion of $\mathcal{U}$ above λ and one of the upper hulls below λ passing through the point nearest to p in $\mathcal{U}$. The points q and r lie in τ.

included in $\mathcal{B}$). The algorithm certifies that $\mathcal{B}$ forms part of the output by computing common tangents between a portion of $\mathcal{U}$ and the other upper hulls. Computing a common tangent between two upper hulls could be costly, but if there is a line separating them, then the time complexity is logarithmic [2]. The algorithm takes advantage of this fact by using as separating lines two tangents through p computed in the previous step (i.e., ignoring the portion of $\mathcal{U}$ in the same half plane as the other upper hulls). The block $\mathcal{B}$ is determined by the common tangents passing through the points nearest to p in $\mathcal{U}$ (one point to the left of p and the other one to the right). To avoid the computation of all common tangents, the algorithm interweaves the different tangent computations (similarly to how Demaine et al.'s algorithm [7] interweaves doubling searches to compute the intersection of sorted sets). We devote the rest of the section to describe this step in more details.

We describe how to identify the part of $\mathcal{B}$ to the right of p (the left counterpart is symmetric). Let λ be the tangent of maximum slope between p and the upper hulls to the right of p (i.e., the tangent of maximum slope among those computed in the previous step of the algorithm). Let $\mathcal{U}'$ be the portion of the upper hull $\mathcal{U}$ containing p above λ. The tangent λ separates $\mathcal{U}'$ from the upper hulls below λ. Among the common tangents between $\mathcal{U}'$ and the upper hulls below λ, let τ be the one passing through the nearest point to p in $\mathcal{U}'$. Let q and r be the points that lie in τ, such that q belongs to $\mathcal{U}'$ and r belongs to one of the upper hulls below of λ. The point q determines the end of the right portion of $\mathcal{B}$ (see Fig. 2 for a graphical representation of these definitions).

Given two upper hulls $\mathcal{A}$ and $\mathcal{B}$ separated by a vertical line, Barbay and Chen [2] described an algorithm that computes the common tangent τ between them, in time within $O(\log a + \log b)$, where a and b are the ranks of the points that lie in τ in the sequences of points representing $\mathcal{A}$ and $\mathcal{B}$, respectively. At each step this algorithm considers two points: one from $\mathcal{A}$ and the other one from $\mathcal{B}$, and in at least one upper hull, it can certify, in constant time, if the point that lies in τ is to the right or to the left of the point considered. A minor variant manages the case where the separating line is not vertical: as the first

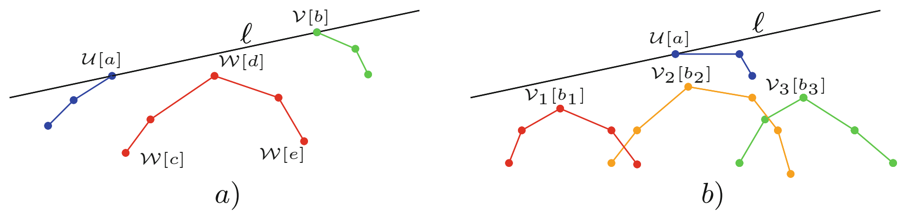

Fig. 3. Example of arguments: (a) an eliminator argument formed by 3 blocks and (b) a convex argument formed by 4 blocks.

step, in each upper hull, the algorithm computes the supporting line of slope equal to the slope of the separating line, by performing doubling searches.

To compute the point q that determines the right portion of $\mathcal{B}$, the algorithm `Quick Union Hull` executes several instances of the algorithm described by Barbay and Chen [2] for computing the common tangents between $\mathcal{U}'$ and the upper hulls below λ, always considering the same point u in $\mathcal{U}'$. Once all decisions about the point u are reached, the upper hulls below λ can be divided into two sets: (i) those whose common tangents pass through a point to the left of u in $\mathcal{U}'$, and (ii) those whose common tangents pass through a point to the right of u in $\mathcal{U}'$. If the set (i) is not empty, then the algorithm stops the computation in the set (ii). For each upper hull $\mathcal{V}$ in the set (ii), the algorithm discards the block of points in $\mathcal{V}$ to the left of the penultimate point considered. This step continues until there is just one instance running, and computes the tangent τ in this instance. The algorithm discards all points to the left of r (i.e., all points that lie below the arc of the output that leaves $\mathcal{U}$ clockwise and follows τ).

After identifying the block $\mathcal{B}$ of the output, the algorithm recurses on the resulting upper hulls to the left and right of p.

2.2 Complexity Analysis of the Quick Union Hull Algorithm

Each algorithm that solves the Union of Upper Hulls problem needs to certify that some blocks of points in the upper hulls cannot participate in the upper hull of the union, and that some other blocks are indeed in the upper hull of the union. In the following, we formalize the notion of partition certificate, which can be used to check the correctness of the output in less time than to recompute the output itself. A *partition certificate* of an instance is a partition of the points of the upper hulls into regions so that, in each region, it is "easy" to certify whether the points form part of the output or not. This notion of partition certificate yields a measure of the difficulty of an instance ("short" partition certificates characterize "easy" instances, while "long" partition certificates suggest "difficult" instances). We define a language of basic arguments for such partition certificates: *eliminator* arguments discard points from the input and *convex* arguments justify the presence of points in the output. A partition certificate is formed by eliminator and convex arguments and will be verified by checking each of its arguments. See Fig. 3 for a graphical representation of such arguments.

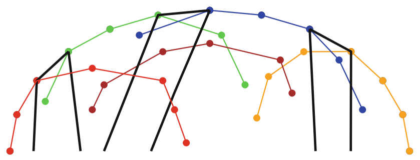

Fig. 4. A partition certificate of size 7 of an instance of the UNION OF UPPER HULL problem. The thick black lines mark the division between the 7 regions.

Definition 1. *Consider the upper hulls $\mathcal{U}$, $\mathcal{V}$, and $\mathcal{W}$. Let ℓ be the straight line through the points $\mathcal{U}[a]$ and $\mathcal{V}[b]$. $\langle \mathcal{U}[a], \mathcal{V}[b] \supset \mathcal{W}[c..d..e]\rangle$ is an* Eliminator Argument *if the points of the block $\mathcal{W}[c..e]$ are between the vertical lines through $\mathcal{U}[a]$ and $\mathcal{V}[b]$, the slope of ℓ is between the slopes of the two edges in $\mathcal{W}$ that precede and follow the point $\mathcal{W}[d]$, and the point $\mathcal{W}[d]$ lies below ℓ.*

If $\langle \mathcal{U}[a], \mathcal{V}[b] \supset \mathcal{W}[c..d..e]\rangle$ is an eliminator argument, then the points of the block $\mathcal{W}[c..e]$ cannot contribute to the upper hull of the union. Several blocks that are "eliminated" by the same pair of points can be combined into a single argument. These eliminator arguments are the ones used in the Steps 4 and 6 of the algorithm `Quick Union Hull`.

It is not enough to discard some points that do not contribute to the output. Certifying still requires additional work: a correct algorithm must justify the exactness of its output. To this end we define convex arguments.

Definition 2. *Consider the upper hulls $\mathcal{U}, \mathcal{V}_1, \ldots, \mathcal{V}_t$. $\langle \mathcal{U}[a] \dashv \mathcal{V}_1[b_1], \ldots, \mathcal{V}_t[b_t]\rangle$ is a* Convex Argument *if there is a straight line ℓ through $\mathcal{U}[a]$ such that the slope of ℓ is between the slopes of the edges that precede and follow the points $\mathcal{V}_1[b_1], \ldots, \mathcal{V}_t[b_t]$, respectively, and the points $\mathcal{V}_1[b_1], \ldots, \mathcal{V}_t[b_t]$ lie below ℓ.*

If $\langle \mathcal{U}[a] \dashv \mathcal{V}_1[b_1], \ldots, \mathcal{V}_t[b_t]\rangle$ is a convex argument, then the point $\mathcal{U}[a]$ is a vertex of the upper hull of the union of $\mathcal{U}, \mathcal{V}_1, \ldots, \mathcal{V}_t$. Blocks of points can also be "easily" certified as part of the output using similar arguments: when the first and last points p and q, respectively, in such blocks are vertices of the output, and all other points in the instance lie below the line through p and q. These convex arguments are the ones used in Step 5 of `Quick Union Hull`.

Those arguments are a two-dimensional generalization of the arguments from Demaine et al. [7] for computing the UNION OF SORTED SETS, and are inspired by the ones introduced by Barbay and Chen [2] for the binary UNION OF UPPER HULLS. Those atomic arguments combine into a general definition of *partition certificate* that any correct algorithm for solving the UNION OF UPPER HULLS problem in the algebraic decision tree model can be modified to output (see Fig. 4 for an example of such a partition certificate).

Definition 3. *Given an instance $\mathcal{I}$ of the* UNION OF UPPER HULL *problem, a* Partition Certificate *of $\mathcal{I}$ is a partition of the points into regions, so that in each region, the points of $\mathcal{I}$ that belong to the output can be decided using a constant number of eliminator and convex arguments. The* Size *of $\mathcal{I}$ is the number of regions which compose it.*

The algorithm `Quick Union Hull` partitions the upper hulls into blocks of consecutive points, where each block is either discarded or output. A block is discarded if it is underneath the upper hull of the union, or is output if it forms part of the upper hull of the union. Each of such blocks forms part of an argument of the partition certificate computed by the algorithm. We separate the analysis of the steps that discard or output blocks of points (i.e., Steps 2, 3, 4, 5, and 6)[3] from the steps that compute the medians of the slopes of the middle edges (i.e., Step 1). The following lemma states that the asymptotic time complexity for discarding a block s is logarithmic in the number of points in s.

Lemma 1. *Given an upper hull $\mathcal{U}$, the cumulated time complexity of the steps that discard blocks of points of the algorithm* `Quick Union Hull` *considering only points of $\mathcal{U}$ is within $O(\sum_{j=1}^{\beta} \log s_j)$, where $s_1, \ldots, s_\beta$ are the sizes of the β blocks into which the whole algorithm partitions $\mathcal{U}$.*

We state the following lemmas in function of the partition certificate computed by the algorithm. The blocks that are discarded in each execution of the Steps 4 and 6 are certified using a single eliminator argument. In the same way, the block that is output in Step 5 is certified using a single convex argument.

Lemma 2. *Given a block $\mathcal{B}$ that forms part of the output, the time complexity of the step that outputs $\mathcal{B}$ of the algorithm* `Quick Union Hull` *is within $O(w \log s)$, where s is the size of $\mathcal{B}$ and w is the number of arguments in the convex argument used by the algorithm to certify that $\mathcal{B}$ forms part of the output.*

This is a consequence of the w searches for the common tangent in Step 5. The amount of arguments in the partition certificate and the number of blocks in each of the arguments are related to the time complexity of Step 1.

Lemma 3. *Given ρ upper hulls, the cumulated time complexity of the steps that compute the medians of the slopes of the middle edges of the algorithm* `Quick Union Hull` *is within $O(\sum_{i=1}^{\delta} \log \binom{\rho}{m_i})$, where δ is the size of the partition certificate $\mathcal{C}$ computed by the algorithm, and $m_1, \ldots, m_\delta$ is a sequence where m_i is the number of blocks in the i-th argument of $\mathcal{C}$.*

We describe an analysis of the algorithm `Quick Union Hull` in function of the smallest possible size δ of a partition certificate for a particular instance.

Theorem 1. *Given ρ upper hulls of sizes $r_1, \ldots, r_\rho$ such that the upper hull of their union admits a partition certificate of size δ, there is an algorithm that computes the upper hull of their union in time within $O(\rho\delta \log \frac{h}{\delta} + \delta \sum_{i=1}^{\rho} \log \frac{r_i}{\delta})$, where h is the number of points in the upper hull of their union.*

Proof. The size of the partition certificate $\mathcal{C}$ computed by the algorithm `Quick Union Hull` in an instance $\mathcal{I}$ is a constant factor of the size δ of a partition

[3] Even though the Steps 2 and 3 do not discard or output blocks of points by themselves we include them in the same analysis as the Steps 4 and 6.

certificate $\mathcal{P}$ of minimal size for $\mathcal{I}$, such that in each region there is just one block that forms part of the output, and this block can be certified using a single convex argument. Indeed, if a region $\mathcal{R}$ of $\mathcal{P}$ contains a block $\mathcal{B}$ that forms part of the upper hull $\mathcal{U}$ of the union, then the algorithm `Quick Union Hull` can certify that $\mathcal{B}$ forms part of $\mathcal{U}$ using a constant number of arguments. This is a consequence of the step that computes the blocks that form part of the output. This step computes the block of maximum size (p included) that can be certified that forms part of the output using a single convex argument. In addition, the algorithm partitions each upper hull in at most a constant factor of δ blocks. Combining the results from Lemmas 1, 2, and 3 with the concavity of the logarithm function, we obtain that the time complexity is within $O(\sum_{i=1}^{\rho}\sum_{j=1}^{\delta} s_{ij} + \sum_{k=1}^{\delta} w_k \log n_k) \subseteq O(\delta \sum_{i=1}^{\rho} \log \frac{r_i}{\delta} + \rho\delta \log \frac{h}{\delta})$, where s_{ij} is the size of the j-th block of the i-th upper hull, w_k is the number of arguments in the k-th convex argument, and n_k is the size of the k-th block of $\mathcal{U}$. $\square$

In the following section, we combine the union algorithm with an algorithm that partitions the sequence of points into "easy" instances, to obtain a synergistic algorithm that computes the convex hull of a set of planar points.

3 Synergistic Computation of Convex Hulls

We describe a synergistic algorithm for computing the convex hull of a set of planar points. It is synergistic in the sense that it takes advantage of both the order of the points and their positions at once, whereas all previous solutions take advantage only of one of those. As a consequence, this algorithm outperforms the best previous solutions [1,10], as well as any dovetailing combination of them. This algorithm decomposes first the input sequence of points into simple subchains (Sect. 3.1), computes their convex hulls [10], and then merges their convex hulls (Sect. 2). There are two noteworthy advantages to this approach: (1) the algorithm decomposes the points into "easy" instances (these "easy" instances are determined by the order in which the points are given), and computes their convex hulls, both steps in time linear in the number of points; and (2) when merging the resulting convex hulls it takes advantage of the number of convex hulls, that the points in the convex hulls are given in sorted order, and the positions of the points (analyzed in Sect. 3.2).

3.1 Linear Time Partitioning Algorithm

A *polygonal chain* is a curve specified by a sequence of points $p_1, \ldots, p_n$. The curve itself consists of the line segments connecting the pairs of consecutive points. A polygonal chain is *simple* if it does not have a self-intersection. Levcopoulos et al. [10] described an algorithm to compute the convex hull of n points in the plane in time within $O(n(1 + \log \kappa))$, where κ is the minimum number of simple subchains into which the input sequence of points can be partitioned. The algorithm tests if the polygonal chain $\mathcal{P}$ given as input is simple:

Algorithm 2. `Doubling Search Partition`

```
Input: A sequence of $n$ planar points $p_1, \ldots, p_n$
Output: A sequence of simple polygonal chains
1: Initialize $i$ to 1;
2: for $t = 1, 2, \ldots$ do
3:    if $i + 2^t - 1 > n$ or the chain $p_i, \ldots, p_{i+2^t-1}$ is not simple then
4:       Output the chain $p_i, \ldots, p_{i+2^{t-1}-1}$
5:       Update $i \leftarrow i + 2^{t-1}$ and $t \leftarrow 1$
```

if $\mathcal{P}$ is simple, it computes the convex hull of $\mathcal{P}$ in time linear in the size of $\mathcal{P}$. Otherwise, if $\mathcal{P}$ is not simple, it partitions $\mathcal{P}$ into two subchains, whose sizes differ at most by one; recurses on each of them; and merges the resulting convex hulls. The time complexity of the partitioning and merging steps are both within $\Theta(n(1 + \log \kappa))$.

We describe an improved partitioning algorithm running in time linear in the size of the input, which is key to the synergistic result. The `Doubling Search Partition` algorithm searches one by one for the largest integer t such that the subchain formed by the first 2^t points is simple. It identifies this subchain as simple and restarts the computation in the rest of the sequence. Its pseudocode is described in Algorithm 2. This algorithm identifies a simple subchain of size k in time within $O(k)$, because the sizes of the tested subchains form a geometric progression of ratio 2. The time complexity of this partitioning algorithm is linear in the number n of points in the sequence, but we prove that the entropy $\mathcal{H}(r_1, \ldots, r_k) = \sum_{i=1}^{k} \frac{r_i}{n} \log \frac{n}{r_i}$ of the sizes $r_1, \ldots, r_k$ of the resulting k simple subchains is a constant factor of the entropy of the sizes of any partition of the sequence of n points into the minimum possible number κ of simple subchains:

Theorem 2. *Given a sequence $\mathcal{S}$ of n planar points, the algorithm `Doubling Search Partition` computes in linear time a partition of $\mathcal{S}$ into k simple subchains of sizes $r_1, \ldots, r_k$, such that $n(1 + \mathcal{H}(r_1, \ldots, r_k)) \in O(n(1 + \alpha))$, where α is the minimum value for the entropy function $\mathcal{H}(s_1, \ldots, s_\kappa)$ of any partition of $\mathcal{S}$ into κ simple subchains, of respective sizes $s_1, \ldots, s_\kappa$.*

Proof. Consider a partition π of $\mathcal{S}$ into κ simple subchains of sizes $s_1, \ldots, s_\kappa$. Fix the subchain c_i of size s_i. The subchain c_i contributes $\frac{s_i}{n} \log \frac{n}{s_i}$ to the value of $\mathcal{H}(s_1, \ldots, s_\kappa)$. The algorithm `Doubling Search Partition` partitions c_i into simple subchains. One of such subchains is at least of size $\frac{s_i}{2}$, and in the worst case, the sizes of the rest of them form a decreasing geometric progression of ratio $\frac{1}{2}$. Hence, the subchains into which the algorithm partitions c_i contribute $O(\sum_{i=1}^{\infty} \frac{s_i}{2^i} \log \frac{2^i n}{s_i}) = O(s_i + \frac{s_i}{n} \log \frac{n}{s_i})$ to the entropy of the partition obtained by the algorithm. The result follows. $\square$

Given the convex hulls of the subchains obtained by the algorithm `Doubling Search Partition`, an algorithm that merges two by two the shortest ones takes advantage of the potential disequilibrium in the distribution of their sizes, a result that improves upon the algorithm described by Levcopoulos et al. [10]:

Corollary 1. *Given a sequence $\mathcal{S}$ of n planar points that can be partitioned into κ simple subchains of respective sizes $r_1, \ldots, r_\kappa$, there is an algorithm that computes the convex hull of $\mathcal{S}$ in time within $O(n(1 + \mathcal{H}(r_1, \ldots, r_\kappa))) \subseteq O(n(1+ \log \kappa))$.*

3.2 Synergistic Algorithm to Compute the Convex Hull

Given a set $\mathcal{S}$ of planar points, the algorithm `Quick Synergy Hull` computes the upper hull of $\mathcal{S}$. It proceeds in two phases. It first partitions $\mathcal{S}$ into simple subchains using the algorithm `Doubling Search Partition` (described in Sect. 3.1), and computes the upper hulls of the simple subchains [10], both steps in time linear in the number of points in $\mathcal{S}$. Then it merges those upper hulls using the algorithm `Quick Union Hull` (described in Sect. 2).

The algorithm `Quick Synergy Hull` outperforms both the algorithm described by Levcopoulos et al. [10] and the one described by Kirkpatrick and Seidel [9] (even when analyzed by Afshani et al. [1]), as well as any dovetailing combination of them. We prove this more formally in the following theorem:

Theorem 3. *Consider a sequence $\mathcal{S}$ of n planar points that can be partitioned into κ simple subchains of sizes $r_1, \ldots, r_\kappa$ (such that $\sum_{i=1}^{\kappa} r_i = n$); and also can be partitioned into h sets of sizes $n_1, \ldots, n_h$ (such that $\sum_{i=1}^{h} n_i = n$), where each set can be enclosed by a triangle completely below the upper hull of $\mathcal{S}$. There is an algorithm that computes the upper hull of $\mathcal{S}$ in time within $O(n + \sum_{j=1}^{\delta} w_j \log s_j + \sum_{i=1}^{\delta} \log \binom{\kappa}{m_i}) \subseteq O(n(1 + \min(\mathcal{H}(r_1, \ldots, r_\kappa), \mathcal{H}(n_1, \ldots, n_h))))$ $\subseteq O(n(1 + \min(\log \kappa, \log h))) \subseteq O(n \log n)$, where the union of the upper hulls of the simple subchains admits a partition certificate $\mathcal{C}$ of minimum size δ (such that $\delta \leq h$), $m_1, \ldots, m_\delta$ is a sequence where m_i is the number of blocks in the i-th argument of $\mathcal{C}$ (such that $m_i \leq \kappa$ for $i \in [1..\delta]$), w_j is the number of arguments in the j-th convex argument of $\mathcal{C}$, and s_j is the size of the j-th block of the output.*

Proof. This result is a consequence of Theorems 1 and 2. For example, if the simple subchains obtained by the partitioning algorithm are all of constant size (i.e., the algorithm cannot take advantage of the order of the points), then the time complexity of the algorithm `Quick Synergy Hull` and the one described by Kirkpatrick and Seidel [9] (as analyzed by Afshani et al. [1]) are asymptotically the same. This algorithm also takes advantage of the positions of the points to improve upon the algorithm described in Corollary 1. $\square$

Example 1. Consider for example the family of instances depicted in Fig. 5: on such instances, the time complexity of the algorithm described by Kirkpatrick and Seidel [9], as refined by Afshani et al. [1], is within $O(n + h \log n)$ (all the points in the sequences $1, 2$ and 3 can be enclosed by a triangle completely below the upper hull of the points, hence $n_1 = \cdots = n_{h-1} = 1$ and $n_h = n - h + 1$ in the formula $O(n(1 + \mathcal{H}(n_1, \ldots, n_h)))$, where $h - 1$ is the number of points in the sequence 4). The time complexity on such instances of the algorithm described by Levcopoulos et al. [10], as refined in Sect. 3.1, is within $O(n + \kappa \log n)$ (the

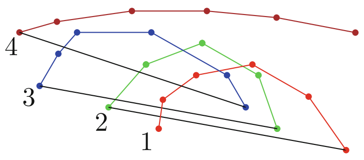

Fig. 5. A sequence of points and its decomposition into $\kappa = 4$ simple subchains. The numbers indicate the order in which the sequence of points are given: each from left to right internally, and mark the simple subchains.

whole sequence of points can be partitioned into κ simple subchains, suppose that the sizes of the simple subchains labeled 1 to $\kappa-1$ are a constant c and that the size of the simple subchain labeled κ is $n-(\kappa-1)c$, then $r_1 = \cdots = r_{\kappa-1} = c$ and $r_\kappa = n-(\kappa-1)c$ in the formula $O(n(1+\mathcal{H}(r_1,\ldots,r_\kappa))))$. On the other hand, the time complexity on such instances of the algorithm `Quick Synergy Hull` is within $O(n)$: once it computes the first vertex of the output, it discards all the points except the points in the upper hull labeled 4. If $h \in \Theta(n)$ and $\kappa \in \Theta(n)$, then the algorithm `Quick Synergy Hull` is faster than the previous algorithms [1,10] by a factor logarithmic in the size of the input.

References

1. Afshani, P., Barbay, J., Chan, T.M.: Instance-optimal geometric algorithms. J. ACM **64**(1), 3:1–3:38 (2017)
2. Barbay, J., Chen, E.Y.: Convex hull of the union of convex objects in the plane: an adaptive analysis. In: Proceedings of the Annual Canadian Conference on Computational Geometry (CCCG) (2008)
3. Barbay, J., Ochoa, C., Satti, S.R.: Synergistic solutions on MultiSets. In: 28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017, Warsaw, Poland, 4–6 July 2017, pp. 31:1–31:14 (2017)
4. Bentley, J.L., Yao, A.C.C.: An almost optimal algorithm for unbounded searching. Inf. Process. Lett. **5**(3), 82–87 (1976)
5. Chan, T.M., Snoeyink, J., Yap, C.K.: Primal dividing and dual pruning: output-sensitive construction of four-dimensional polytopes and three-dimensional voronoi diagrams. Discrete Comput. Geom. (DCG) **18**(4), 433–454 (1997)
6. Chand, D.R., Kapur, S.S.: An algorithm for convex polytopes. J. ACM **17**(1), 78–86 (1970)
7. Demaine, E.D., López-Ortiz, A., Munro, J.I.: Adaptive set intersections, unions, and differences. In: Proceedings of the 11^{th} ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 743–752 (2000)
8. Graham, R.L.: An efficient algorithm for determining the convex hull of a finite planar set. Inf. Process. Lett. **1**, 132–133 (1972)
9. Kirkpatrick, D.G., Seidel, R.: The ultimate planar convex hull algorithm? SIAM J. Comput. (SICOMP) **15**(1), 287–299 (1986)
10. Levcopoulos, C., Lingas, A., Mitchell, J.S.B.: Adaptive algorithms for constructing convex hulls and triangulations of polygonal chains. In: Penttonen, M., Schmidt, E.M. (eds.) SWAT 2002. LNCS, vol. 2368, pp. 80–89. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45471-3_9