Adaptive (Analysis of) Algorithms for Convex Hulls and Related Problems.

Jérémy Barbay

Departamento de Ciencias de la Computación (DCC),
Universidad de Chile, Santiago, Chile.
`jbarbay@dcc.uchile.cl`

Technical Reporte TR/DCC-2008-17
received on December 2nd, 2008
by the Departamento de Ciencias de la Computación (DCC),
Universidad de Chile, Santiago, Chile.
`http://www.dcc.uchile.cl/1877/propertyvalue-11449.html`

Abstract. Adaptive analysis is a well known technique in computational geometry, which refines the traditional worst case analysis over all instances of fixed input size by taking into account some other parameters, such as the size of the output in the case of output sensitive analysis. We present two adaptive techniques for the computation of the convex hull in two and three dimensions and related problems. The first analysis technique is based on the *input order* and yields results on the computation of convex hulls in two and three dimensions, and the first adaptive algorithm for Voronoi and Delaunay diagrams, through the entropy of a partition of the input in easier instances. The second analysis technique is based on the *structural entropy* of the instance, and yields results on the computational complexity of planar convex hull and of multiset sorting, through a generalization of output sensitivity and a more precise analysis of the complexity of Kirkpatrick and Seidel's algorithm. Our approach yields adaptive algorithms which perform faster on many classes of instances, while performing asymptotically no worse in the worst case over all instances of fixed size.

1 Motivations

Worst Case Computational Complexity. The field of "Computational Complexity" in theoretical computer science, and more particularly in computational geometry, is all about describing the complexity of *problems* rather than algorithms. Traditionally, two types of complexities have been studied: worst case and average case: we focus in this paper on the first one. Given a problem P and a positive integer n, the worst case computational complexity of P over instances of size n is the worst performance $f(n)$ of the best algorithm possible over all instances of size n. The positive integer function f implied by this definition is the "worst case computational complexity of P", with the underlying assumption that each worst case is chosen among instances of similar sizes.

Restricting the study to the set of instances of sizes n (or, without lack of generality, of sizes at most n) reduces the analysis of the worst case complexity of a deterministic algorithm A to the maximum of a finite set, the set of the complexities of A over each instance of size n. In turn, it reduces the analysis of the worst case complexity of a problem P to the computation of the minimum of a maximum, leading to lower bound techniques and tight bounds on the computational complexity. The concept is easily generalize to problems where the input size is not easily defined by a single parameter (e.g. graphs with n vertices and m edges, or multisets of n numbers from $[\sigma] = \{1, \ldots, \sigma\}$).

The same approach can be taken with a finer partitioning of the set of possible instances in finite subsets. *Output sensitive* results correspond to the application of this principle by separating further the instances by the size h of their corresponding solution. This approach yields interesting results for the computation of the convex hull in two and three dimensions, and for related problems in computational geometry [8, 23]. Moreover, this type of analysis is a requirement to compare the performances of pattern matching [32] and range searching data structures [1], in which the query time is often of the form $O(f(n)+h)$, where h is the number of matches satisfying the query, i.e. the output size. Some problems present instances which are still of distinct difficulty while of same input and output size. Since any correct algorithm must check the correctness of the solution it outputs, one can require that each algorithm outputs a *certificate* of its solution in addition or instead of its solution, permitting once again to perform an output sensitive analysis of the computational complexity of the problem. This approach yields interesting results on the computation of the *description* of the union of sorted sets [11], on the computation of a *certificate* of the intersection of sorted arrays [4] or of the intersection of planar convex hulls [3]. In those cases, measuring the size of the output in bits [11] or in words [3, 4] yields different kind of analysis.

This technique can be applied even to NP-hard problems for which there is no computational lower bound, via reductions between problems and the definition of classes of equivalences. For instance, in some NP-hard problems each instance can be reduced in polynomial time to a "core" instance of smaller size (say k), so that an upper bound on the cost to solve the original instance is exponential in k but polynomial in n. Under the name of "Parameterized Complexity" [16, 30], this approach yields interesting results and in particular a classification theory of computational problems which is more useful in practice than the polynomial hierarchy.

Input Order Adaptivity. All the previous examples can be seen as some sort of output sensitivity, in some cases through redefining the output to include a certificate of correctness. Another approach is to take advantage of the encoding of the input for those problems where the input can be encoded in several distinct ways, and in particular where the input can be encoded in several orders.

One example of computational problem in this category is sorting: a set is more easily sorted when it is input as an array already sorted or "almost sorted", as in this case a linear algorithm suffices to check the order and perform minor corrections [25]. Whereas any sorting algorithm in the comparison model requires time $\Omega(n \lg n)$ to sort n values in the worst case[1], several improvements have been demonstrated on more specific classes of instances, such as instances formed by permutations formed of a small number of sorted blocks [29] (e.g. $(\mathit{1}, \mathit{3}, \mathit{5}, \mathit{7}, \mathit{9}, \mathbf{2}, \mathbf{4}, \mathbf{6}, \mathbf{8}, \mathbf{10})$ or $(\mathit{6}, \mathit{7}, \mathit{8}, \mathit{9}, \mathit{10}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5})$) , or permutations containing sorted subsequences [27] (e.g. $(\mathit{1}, \mathbf{6}, \mathit{2}, \mathbf{7}, \mathit{3}, \mathbf{8}, \mathit{4}, \mathbf{9}, \mathit{5}, \mathbf{10})$) . Algorithms performing $o(n \lg n)$ comparisons on such permutations, yet still $\mathcal{O}(n \lg n)$ comparisons in the worst case over all instances of size n, are achievable and obviously preferable: see Mannila's seminal paper [29] and Estivil-Castro and Wood's review [14] for more details, formal reductions between measures of disorder, and additional bibliography.

Given a finite set S of points, its *convex hull* [34] is the smallest convex polyhedra containing S. Computing the convex hull is equivalent to computing the lower and *upper hull*, as the convex hull is the union of both, and one can deduce both from the convex hull. If the points are planar, we consider its *Voronoi diagram* [35], defined as a a partition of the plane in regions of closest neighborhood for S, and its *Delaunay diagram*, the dual partition of the Voronoi diagram of S. The computation of the convex hull is a central problem in Computational Geometry, and was one of the problems at the origin of the field [34, 35]. It has many applications, in particular to the computation of Delaunay and Voronoi diagrams. The problem of computing the two and three dimensional convex hull, of computing the Delaunay and Voronoi diagrams, and the problem of sorting, are all closely connected in a chain of linear time reductions: any algorithm computing three dimensional convex hulls can be used to compute Delaunay diagrams, which envelope is the two dimensional convex hull, which in turn can be used to sort values. The convex hull of n points in two and three dimensions can be obtained in output sensitive time $\Theta(n(1 + \lg h))$, where h is the number of edges or faces which form the output. This yields a worst case complexity of $\Theta(n \lg n)$ for the computation of Delaunay and Voronoi diagrams, via a simple reduction mapping a set of planar input points to a convex surface in three dimension [35]. Since the computational complexity of the planar convex hull is linear when the input is sorted by abscissa, there is a direct relation between sorting and the computation of planar convex hulls. Various "good" input orders are known for the computation of Delaunay and Voronoi diagrams (and hence of planar convex hulls). If the input points are ordered so that they form a simple [9, 35] or convex [2, 24] polygon, or a monotonic histogram [12], then their Delaunay and Voronoi diagrams can be computed in linear time. This shows the existence of "easy" instances, yet very few have tried to exploit those instances formally, and the potential existence of "almost easy" instances. Levcopoulos *et al.* [26] defined an algorithm computing the planar convex hull which is adaptive to the number χ of maximal simple subchains in the sequence of input points, with a complexity of $\mathcal{O}(n \lg(\chi + 2))$. They also showed a much weaker result, that a polygonal chain with κ proper intersection points can be transformed into a polygonal chain without proper intersections by adding $\mathcal{O}(\kappa)$ new vertices in time $\mathcal{O}(n \times \min\{\sqrt{\kappa}, \lg n\} + \kappa)$, hence yielding an algorithm computing the Delaunay diagram in this time.

[1] The function $\lg(x)$ denotes the logarithm in base two, and the expression $[x]$ denotes the set $\{1, \ldots, x\}$.

Challenges. The worst case computational complexity over instances of size n is the same $\Theta(n \lg n)$ for a large family of problems, from sorting permutations to computing the convex hull of n points in three dimensions, including the sorting of multisets, the computation of the convex hull in two dimensions, the computation of Voronoi and Delaunay diagrams, the computation of *kd* trees, the computation of continuous skeletons, etcetera. Given that we gained a better understanding of some of those problems (e.g. sorting and computing the convex hull in two dimensions) through adaptive analysis, it is only natural to try to gain a better of the other problems as well.

The worst case computational complexity over instances of fixed size of those problems are reducible one to another, because sorting is a particular case of sorting multisets, which is a particular case of the computation of a two dimensional convex hull, itself a particular case of the computation of a Delaunay diagram, which can be performed through any algorithm computing the convex hull in three dimensions. But those reductions do not imply similar ones on the worst case computational complexity over restricted classes of instances. In particular, Seidel [36] showed that, contrarily to the computation of the planar convex hull, sorting points in three dimensions by their coordinates does not help to compute their Delaunay diagram nor their convex hull, and conversely that knowing one of their triangulation (even if it is their Delaunay triangulation, more constrained and hence holding more information) or their convex hull in three dimensions, does not help to sort their coordinates in general.

Hence the question holds of defining adequate adaptive analysis for each of these problems, and eventually reduce between themselves the pairs formed by problems and adaptive analysis, in the spirit of common reduction techniques from parameterized complexity. This means not only defining some adaptive analysis for the computation of Delaunay and Voronoi diagrams, for which no such analysis currently exists, but also providing new adaptive analysis techniques for the other problems, in order to get a better understanding of their computational complexity.

Our Results. We give a more precise analysis of the asymptotic computational complexity of convex hulls in two and three dimensions through two new measures of difficulty, showing for each upper and lower bounds and hence providing more precise information on the computational complexity of those problems than the traditional worst case analysis over all instances of fixed size.

Our first measure of difficulty depends on the *input order*, through the entropy of a partition of the input in subsequences corresponding to easier instances. We describe in Section 2 how to partition a three dimensional instance of the convex hull into easier ones, and how to merge efficiently the solutions of those sub-instances by taking advantage of their relative sizes. This result implies the existence of adaptive algorithms for many other problems than the computation of convex hulls in two or three dimensions, including in particular the computation of Delaunay and Voronoi diagrams [35] and sorting of multisets. The projection of our adaptive analysis to the computation of the planar convex hull is independent of both the number χ of maximal simple subchains and of the number κ of proper intersection points in the sequence of input points, as defined by Levcopoulos *et al.* [26]: our results are not reducible to theirs, nor are their results reducible to ours. The projection of our adaptive analysis to sorting is more precise than the number of "down-steps" in the permutation [25], independent of the entropy of the associated partition of the instance [5], and of all the measures of disorder (other than the number of down-steps) described in the survey from Estivil-Castro and Wood's review [14].

Our second measure of difficulty is the *structural entropy*, which measures the repartition of the planar points of the input relatively to their upper hull, relatively to their projection on the abscissa axis. We show in Section 3 that the original algorithm from Kirkpatrick and Seidel [23] is in fact optimally adaptive to the entropy of the geometric repartition of the input points relatively to the edges of the upper hull. Additionally, this analysis permits to separate the performance of Kirkpatrick and Seidel's algorithm from the performance of the algorithm proposed later by Chan [8], which is optimally output sensitive (as Kirkpatrick and Seidel's) but is *not* adaptive to the structural entropy. This measure of difficulty is completely independent of the order in which the points are input, and hence independent from our other results.

Our approach yields algorithms which are asymptotically faster on many classes of instances, while asymptotically no worse than traditional algorithms when considering the worst case over instances of fixed size. In the spirit of other computational geometry algorithms [31], each of our algorithms produces a certificate of its output, which can be used to check the validity of the computation or to update the solution when the points in the input are moved. We use succinct data structure techniques to insure that the length of those certificates is shorter than a mere description of the computation itself for most instances, and that those description are navigable in reasonable time. Our results are based on definitions and techniques from various fields, such as computational geometry for the partitioning of instances in easier ones [17, 28, 37], the merging of convex hulls [10, 22], the notion of certificates of the results [31]; succinct data structures for the succinct encoding of certificates [5]; adaptive sorting algorithms for the definition of difficulty measures, adaptive lower bounds and adaptive reductions [14, 25, 29]; parameterized complexity for the definition of reductions between pairs of problem and parameters [16]; and compression theory for the design of optimal merging schemes [18, 19].

2 Input Order

The first measure of difficulty that we study is the entropy of a partition of the input in easier instances, based on the order in which the input is given. We describe in Section 2.1 how to partition an instance in easier ones, how to merge the solutions of those sub-instances adaptively in Section 2.2, and some combinatoric results showing the adaptive optimality of the resulting algorithms in Section 2.4.

2.1 Partitioning

Various "good" input orders are known for the computation of the Delaunay triangulation: if the input points are ordered so that they form a simple [9, 35] or convex [2, 24] polygon, or a monotonic histogram [12], then their Delaunay and Voronoi diagrams can be computed in linear time. In a more general approach, Snoeyink and Kreveld [37] described how to encode a Delaunay diagram in the *order* of the points, so that it can be decoded in linear time. Whereas those results seem *a priori* more related to the encoding of triangulations rather than to their computation, their decoding algorithm is in fact computing the Delaunay diagram in linear time, suggesting the existence of an adaptive algorithm, which complexity would gradually degrade from linear to worse time depending of how distant the input order is from one of those defined by Snoeyink and Kreveld [37]. Denny and Sohler [28] generalized Snoeyink and Kreveld's work [37] to the encoding of arbitrary triangulations. Since there is much more flexibility in the edges chosen for a particular triangulation (not as constrained as a Delaunay triangulation), their encoding requires that the instances is at least of size 1090 to be able to encode the required information in the order of a subset of the points. Trivially, Denny and Sohler's result could be used to encode the specific triangulation corresponding to the projection in two dimensions of a three dimensional upper hull, although their encoding has a lot of complications that we don't need when we have the z-coordinate of each point.

We define a linear best case algorithm computing the Voronoi and Delaunay diagrams using the edge algebra defined by Guibas and Stolfi [17]. This algorithm can be used to detect if the input is in an order which is not optimal, which is not explicitly the case in Snoeyink and Kreveld or Denny and Sohler's decoding algorithms. From there, it is quite simple to partition the input into smaller segments, for each of which the Delaunay diagram can be computed in linear time. More generally, the two operators defined by Guibas and Stolfi [17] also permit to represent and build the (general) triangulation representing the projection of three dimensional upper hulls, as they support the addition of a point p, the (adaptive) location of the projection F' of the face F in which the projection p' of p belongs, and the update and removal of the newly covered faces if p is above F.

We show that the existence of these good orders, which can be checked in linear time, implies the existence of partitions of instances in easier one, and of linear time algorithms computing those partitions. In this section and the following one, we describe the details of this approach for the most general problem, the computation of convex hulls in three dimensions. We explore how a similar approach can be applied to other problems such as the computation of the convex hull in two dimensions and sorting in Section 2.3.

Definition 1. *Consider some non negative integer $c \geq 0$, and a sequence S of points in three dimensions. The sequence $S = (x_1, \ldots, x_n)$ is* c-progressive *if each point x_i after the three first ones satisfies the following conditions at the moment of its insertion:*

1. *x_i is separated from its predecessor by at most c edges in the triangulation corresponding to the projection of the upper hull of $(x_1, \ldots, x_{i-1})$ on the horizontal plane; and*
2. *at most c faces need to be corrected between the upper hull of $(x_1, \ldots, x_{i-1})$ into the upper hull of $(x_1, \ldots, x_i)$.*

This definition of c-progressive sequences capture the common notion between the two linear encodings of triangulation mentioned above. The sequence of planar points in the order described by Snoeyink and Kreveld [37] to encode their Delaunay diagram is a c-progressive sequence for the computation of the convex hull in two dimensions. The sequence of points in three dimensions in the order described by Denny and Sohler [28] to encode the triangulation corresponding to the projection of their upper hull on the horizontal plane, is a c-progressive sequence for the computation of the convex hull in three dimensions. Furthermore, the traversal of a complete binary search tree, level by level, from right to left on odd levels and from left to right on even levels, is a 3-progressive sequence for sorting.

More specifically, Snoeyink and Kreveld [37] showed that any point sequence can be reordered into a c-progressive sequence for the computation of Delaunay diagrams, hence showing the existence of c-progressive sequences. Denny and Sohler's work [28] shows the existence of c-progressive sequences for arbitrary triangulations, and hence for the computation of upper hulls of three dimensions, which we exploit in the following lemma:

Lemma 1. *Given a sequence S of n points in three dimensions and a non negative integer $c \geq 0$, there is an algorithm which partitions optimally S into c-progressive subsequences in time $\mathcal{O}(cn)$, and which computes the convex hull of those subsequences at the same time.*

Proof. (of Lemma 1) By definition, the *insertion* algorithm presented by Guibas and Stolfi [17] does not need to cross nor correct more than c edges for each new point considered, in the original input order, on a c-progressive subsequence. Hence it suffices to interrupt the incremental algorithm every time that it performs more than c applications of the two operators defined by Guibas and Stolfi [17] for a single insertion, and reset it on the remaining points of the sequence. $\square$

The partitioning algorithm is already given by the definition of c-progressive sequences. The interesting part concerns how to efficiently merge the sub-solutions thus defined and built, which we describe in the next section.

2.2 Merging

Once the input is partitioned into ρ subsequences for which the problem has already been solved, one can trivially merge them two by two in time $\mathcal{O}(n \lg \rho)$ using Kirkpatrick's algorithm in two dimensions or Chazelle's algorithm in three dimensions, which yields an algorithm of complexity $\mathcal{O}(n(c + \lg \rho))$. This result can be improved to take advantage of the cases where the convex hulls of the sub-instances differ greatly, by using a merging scheme inspired by the Huffman tree from coding theory [19].

Lemma 2. *The union of ρ convex hulls in three dimensions, of respective sizes* $\texttt{Runs} = (n_1, \dots, n_\rho)$ *summing to* $n = \sum_{i\in[\rho]} n_i$*, can be computed in time* $\mathcal{O}(n(1 + \mathcal{H}(\texttt{Runs})) + \rho \lg \rho) \subseteq \mathcal{O}(n(1 + \lg \rho)) \subseteq \mathcal{O}(n \lg n)$*, where* $\mathcal{H}(\texttt{Runs}) = \sum_{i\in[\rho]} \frac{n_i}{n} \lg \frac{n}{n_i}$ *is the entropy of the size vector* $\texttt{Runs}$*.*

Proof. (of Lemma 2) Given the ρ frequencies of a set of symbols, Huffman [19] described how to construct optimal codes for those frequencies in time $\mathcal{O}(\rho \lg \rho)$. Given the sizes $\texttt{Runs} = (n_1, \dots, n_\rho)$ of the Delaunay diagrams, the same technique can be used to produce in time $\mathcal{O}(\rho \log \rho)$ an optimal binary merging schedule. This schedule is such that $L = \sum \ell_i n_i$ is minimal and smaller than $n\mathcal{H}(\texttt{Runs})$, where ℓ_i is the number of times the i-th sub-instance is merged, in a similar way to how a Huffman tree [19] optimizes the code bit of a symbol in function of its frequency.

This binary merging schedule can then be performed using linear time merging algorithms to merge each pair of convex hulls of sizes n_1 and n_2 in time $\mathcal{O}(n_1 + n_2)$, with an output of size at most $n_1 + n_2$. One can use Chazelle's algorithm [10] to merge three dimensional convex hulls, Kirkpatrick's algorithm[22] to merge Voronoi and Delaunay diagrams, or the traditional merge algorithm on sorted arrays to merge two dimensional convex hulls. The optimality of the schedule means that the i-th initial sub-instance of size n_i will contribute at most $\ell_i n_i$ operations to the complexity of the merging. By the properties of our merging schedule, the total complexity is $\mathcal{O}(n(1 + \mathcal{H}(\texttt{Runs})) + \rho \lg \rho)$, which is included in the class of asymptotic complexities $\mathcal{O}(n(1 + \lg \rho))$ by convexity of the logarithm function, itself trivially included in the class of asymptotic complexities $\mathcal{O}(n \lg n)$. $\square$

Using a more complex coding technique than the Huffman tree, we can output a short encoding of the certificate of the convex hull computed, which can be used to check the validity of the output faster than by recomputing it in the spirit of the algorithms from the LEDA library [31], or which can be used to maintain the convex hull adaptively when the points are moved.

Corollary 1. *A succinct representation of the certificate of the output of the algorithm described in Lemma 2 can be encoded in* $2(n\mathcal{H}(\texttt{Runs}) + \sum_{i\in[\rho]} \lg n_i + \rho)$ *bits.*

Proof. (of Corollary 1) Given the frequencies of a sequence of distinct symbols, Hu-Tucker's algorithm [18] describes how to construct optimal codes for those frequencies in time $\mathcal{O}(\rho \lg \rho)$, with the additional property that the lexicographic order of the codes produced matches the original order of the symbols. Replacing Huffman's algorithm by Hu-Tucker's algorithm [18] hence permits to describe succinctly all the merging operations performed.

The space then taken by the certificate is the sum of the encodings for the sizes of the sub-instances, using $2\sum_{i\in[\rho]}\lg n_i$ bits (using a gamma code for each length, which codes in unary the logarithm of the number to code, followed by the number itself in unary); the binary merging scheme, using 2ρ bits (it is a binary tree with ρ leaves); the description of the mergings themselves in $2n\mathcal{H}(\texttt{Runs})$, using two bits for each comparison, coding for the four following cases: (1) the point from the first input is output, (2) the point from the second input is output, (3) the point from the first input is skipped, (4) the point from the second input is skipped. The sum of those spaces yields the result. □

Note that one can navigate and search in the certificate as described in the proof of Corollary 1 by simply indexing it using well known succinct data-structures [15, 20], which would use at most $o(n\mathcal{H}(\texttt{Runs})+\sum_{i\in[\rho]}\lg n_i+\rho)$ bits, asymptotically in n.

2.3 Reduction from Sorting

Obviously, the convex hull in two dimensions is a particular case of convex hull in three dimensions. In a similar way, sorting can be seen as a particular case of computation of the planar convex hull, since it is easy to define an order on the points of a planar convex hull.

Lemma 3. *Given a positive integer $c\geq 0$, any algorithm computing the convex hull of an instance S of n points in three dimensions composed of ρ c-progressive subsequences of respective sizes* $\texttt{Runs}=\{n_1,\ldots,n_\rho\}$ *in time $f(n,\mathcal{H}(\texttt{Runs}))$ can be used*

- *to compute the convex hull of an instance S of n planar points composed of ρ c-progressive subsequences of respective sizes* $\texttt{Runs}=\{n_1,\ldots,n_\rho\}$ *in time $\mathcal{O}(n)+f(n,\mathcal{H}(\texttt{Runs}))$; and*
- *to sort a sequence of n values composed of ρ c-progressive subsequences of respective sizes* $\texttt{Runs}=\{n_1,\ldots,n_\rho\}$ *in time $\mathcal{O}(n)+f(n,\mathcal{H}(\texttt{Runs}))$.*

Proof. (of Lemma 3) One dimensional instances can be mapped to space by mapping each value x to a point of coordinates $(x,0,0)$, but then sorting those values is a degenerate case of the computation of the convex hull in three dimensions. Better, we map each value x to a point in the plane of coordinates (x,x^2), and that planar point to a point in three dimensions of coordinates (x,x^2,x^2+x^4). The projection on the horizontal plane of the convex hull of the points in three dimensions yields the convex hull of the planar points, while the projection of this convex hull on the abscissa axis yield the order of the values. □

The decomposition of a permutation (or multiset) into c-progressive subsequences is exactly its decomposition in increasing subsequences defined by Mannila [29] (among others) for $c=1$, but is much more general and interesting for values even as small as $c=3$. The partition into 3-progressive subsequences corresponds to a new type of partitioning: each subsequence corresponds to the level traversal of a semi-complete binary search tree, which can be constructed in linear time using at much three comparisons for the insertion of each element (see Figure 1 for an example).

2.4 Optimality

We show the adaptive optimality of our approach for sorting, which by the adaptive reduction shown in the previous section implies the adaptive optimality of our approach for the convex hull in two and three dimensions. Given two positive integers n and m such that $n \leq m$ and two sets of positive integers $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_m\}$, we say that X is *at most* Y if there is a permutation π over $[m] = \{1, \ldots, m\}$ such that $x_i \leq y_{\pi(i)}\ \forall i \in [n]$.

Lemma 4. *For any positive integer $n > 0$ and any comparison based sorting algorithm A (randomized or deterministic), there is an instance composed of at most ρ c-progressive subsequences of respective sizes at most $\texttt{Runs} = \{n_1, \ldots, n_\rho\}$, over which A performs $\Omega(n(1 + \mathcal{H}(\texttt{Runs})))$ comparisons.*

Proof. (of Lemma 4) This is a simple adaptive reduction [29] from sorting adaptively to the entropy of a permutation in ρ consecutive increasing subsequences of respective sizes at most $\texttt{Runs}' = \{n_1, \ldots, n_\rho\}$, for which Barbay and Navarro [5] proved a computational lower bound of $\Omega(n(1 + \mathcal{H}(\texttt{Runs}')))$ □

Of course, via the reductions defined in the previous section, this lower bound propagates to the computation of the convex hull in two and three dimensions, as well as to the computation of the Delaunay and Voronoi diagrams.

Corollary 2. *For any positive constant $c > 0$ and integer $n > 0$ and any three dimensional convex hull algorithm A (randomized or deterministic) in the algebraic decision tree model, there is an instance composed of at most ρ c-progressive subsequences of respective sizes at most $\texttt{Runs} = \{n_1, \ldots, n_\rho\}$, over which A performs $\Omega(n(1 + \mathcal{H}(\texttt{Runs})))$ operations.*

Proof. (of Corollary 2) This is a simple parameterized complexity reduction [16] from sorting adaptively to the entropy of partitions in c-progressive subsequences, for which Lemma 4 proves a computational lower bound of $\Omega(n(1 + \mathcal{H}(\texttt{Runs})))$ □

Combining the main results of this section yields the following Theorem:

Theorem 1. *Given a positive integer $c \geq 0$, and a sequence S of n points in three dimensions composed of ρ c-progressive subsequences of respective sizes $\texttt{Runs} = \{n_1, \ldots, n_\rho\}$, there is an algorithm computing the convex hull of S in time $\mathcal{O}(n(c + \mathcal{H}(\texttt{Runs}) + \rho \lg \rho) \subseteq \mathcal{O}(n(c + \lg \rho)) \subseteq \mathcal{O}(n(c + \lg n))$, where $\mathcal{H}(\texttt{Runs}) = \sum_{i \in [\rho]} \frac{n_i}{n} \lg \frac{n}{n_i}$ is the entropy of the size vector $\texttt{Runs}$. When c is chosen constant, this is asymptotically optimal over all such instances, among all randomized or deterministic algorithms in the algebraic decision tree model.*

Beside the fact that we consider the convex hull in three dimension while Levcopoulos *et al.* [26] considered the adaptive computation of the convex hull in two dimensions, it is important to note that the projection of our measure of difficulty for the computation of the planar convex hull is independent of both the number χ of maximal simple subchains and of the number κ of proper intersection points in the sequence of input points, as defined by Levcopoulos *et al.* [26], for the simple reason that there are some simple chains which are not c-progressive, and that for some finite c there are some c-progressive chains which are not simple. This means that our results are not reducible to theirs, nor are their results reducible to ours.

The definition of c-progressive sequences can easily be extended to *amortized c-progressive* sequences, where points which are inserted very quickly yield some "credit" which can be used later to insert more difficult points, and points deleting points previously inserted further increase this credit. Such sequences form instances which can still be solved in linear time, and generalize in the plane "simple chains" used by Levcopoulos *et al.* [26]. Additionally, they are well defined in three dimensions, whereas it is harder to define "simple" chains in three dimensions. Our approach still applies to those sequences without any modification, but we prefer the more intuitive notion of c-progressive sequences.

3 Structural Entropy

The adaptivity seen in the previous section is based on an astute division of the instance in smaller ones which are so easy that they can be solved in linear time. Additionally, we saw how one can take advantage of their relative sizes to merge them faster. Another adaptivity technique is to divide the instance arbitrarily (e.g. in two) into chunks of similar size (e.g. $n/2$) and to study how the difficulty of the original instance impacts the difficulty of the chunks thus obtained. We explore such an approach in this section, where we divide the input into chunks of similar size, and consider how the number of faces of the convex hull divides between those faces. In the worst case there are as many convex hull edges in each chunk, which results in upper and lower bounds of $\Theta(n \lg h)$ on the computational complexity of planar upper hull [23]. But by definition the instances where the imbalance is stronger are easier: measuring this imbalance yields a more precise analysis of the computational complexity of the problem, and in particular of the complexity of Kirkpatrick and Seidel's algorithm [23].

3.1 Marriage before Conquest

Kirkpatrick and Seidel [23] proposed the first optimally output sensitive planar convex hull algorithms, which computes the h edges of the upper hull of n planar points in time $\mathcal{O}(n(1 + \lg h)) \subseteq \mathcal{O}(n \lg n)$, so that the convex hull is obtained by unifying the upper and lower hulls. They introduced through this algorithm the formal concept of output sensitivity: Jarvis [21] had already proposed an algorithm to compute the convex hull in $\mathcal{O}(nh)$ orientation tests, but he was advertising it as of complexity $\mathcal{O}(n^2)$. Nielsen and Yvinec [33] generalized Kirkpatrick and Seidel's technique to the convex merging of convex objects, and Edelsbrunner and Shi [13] generalized it to to the computation of the convex hull of points in three dimensions, thus yielding a $\mathcal{O}(n \lg^2 h)$ algorithm to compute the convex hull in three dimensions, which is not optimal but is output sensitive. Chan [8] later improved this last result to the optimal complexity of $\Theta(n(1 + \lg h))$ in two and three dimensions, through a different paradigm based on Jarvis [21]'s algorithm.

The common intuition between the algorithms from Kirkpatrick and Seidel [23], Nielsen and Yvinec [33] and Edelsbrunner and Shi [13] is to focus on the upper hull, and to perform a kind of reversal of the *divide-and-conquer* paradigm previously used to compute the convex hull [34], which Kirkpatrick and Seidel named *marriage-before-conquest*. While a divide-and-conquer algorithm splits the input into chunks of similar sizes, recursively solves them and then merges them; a marriage-before-conquest algorithm splits the input into chunks, *computes the bridge linking them*, and then only recurses in the reduced chunks. This inversion of the basic order of the operations is what yields the output size sensitivity. Since each recursive call yields one edge of the output, there are at most h recursive calls in total. Since each recursive call reduces the size of the problem by a constant c

(e.g. $c = 1/2$ in the original algorithm from Kirkpatrick and Seidel, and $c = 3/4$ for the algorithm from Edelsbrunner and Shi), the subproblems considered at the i-th level of the recursion are of size at most $n * c^i$, summing to $\mathcal{O}(n)$. The three analysis are all based on the fact that the worst case occurs when there are exactly c^i subproblems in each level of recursion up to the level $\lceil \lg h \rceil$: for this worst case, there are at most $\lceil \lg h \rceil$ levels of recursion and at most n points considered within each level. In two dimensions, where the cost of each recursion is linear, it implies that the total running time of the recursion is $\mathcal{O}(n \lg h)$, yielding $\mathcal{O}(n(1 + \lg h))$ when taking into account the linear preprocessing. In three dimensions, where the cost of each recursion is $\mathcal{O}(n(1+\lg h))$ because the algorithm uses the planar algorithm as a black box, it implies that the total running time of the recursion is $\mathcal{O}(n(1 + \lg^2 h))$.

The worst (output sensitive) case analysis presented above makes two assumptions, which study permits to refine the analysis. The first assumption is that no point is eliminated by the computation of a bridge before the last level of recursion. This supposes that the points which are not part of the upper hull are very precisely positioned below half of the faces of the upper hull, which are exactly alternating with the ones covering no other points below them. The second assumption is that the chunks of equal sizes at each recursion step yield the same number of faces of the upper hull. Put in other terms, it supposes that each face which eliminates points will eliminate on average the same amount of points. The combination of those two assumptions describes worst case instances which are extremely regular, where points which are not part of the upper hull are uniformly distributed below half of the faces of the upper hull, and any instance which does not respect those assumptions will be "easier" for the marriage-before-conquest algorithms. We describe in the following sections a measure of "difficulty" for the computation of the planar upper hull. Our analysis is based on the entropy of the repartition of the points not contributing to the upper hull, relatively to the edges of the planar upper hull, which yields a more precise analysis of the complexity of the marriage-before-conquest algorithms. The same analysis can be performed in three dimensions, but the results it yields are not competitive

3.2 Structural Entropy

Consider a set S of n planar points, among which h form the upper hull, noted $(p_1, \ldots, p_h)$ where the points are ordered by their abscissas. We count for each edge (p_i, p_{i+1}) of the upper hull the number n_i of points of the input which are in the "shadow" of this edge, i.e. which projection on the abscissa axis is contained in the projection of this edge.

Definition 2. *The* Structural Entropy $\mathcal{H}^*(S)$ *of S is the entropy of the repartition of the abscissa of its points among the projected edges of its upper hull:* $\mathcal{H}^*(S) = \sum_{i \in [h]} \frac{n_i}{n} \lg \frac{n}{n_i}$, *where the counters* $\forall i \in [h]$ *are defined by* $n_i = \#\{p \in S \text{ s.t. } p_i.x \leq p.x < p_{i+1}.x\}$ *(placing a virtual point p_{h+1} infinitely to the right in order to simplify the expressions). For ease of notation, we note* $\texttt{Rep}(S) = (n_1, \ldots, n_h)$ *the* repartition *of S.*

In the worst case over instances of upper hull of size h, each edge "covers" roughly the same number of points and the structural entropy within a constant term of $\lceil \lg h \rceil$. But in other cases the structural entropy is much smaller than $\lceil \lg h \rceil$, and the algorithm from Kirkpatrick and Seidel takes automatically advantage of any large disequilibrium (i.e. by at least a constant factor) of the repartition of the points.

Note that the structural entropy is refining the output size, but can also be considered independently from it: there exists pairs (A, B) of instances such that A has a larger output size but smaller structural entropy than B, in a similar way than some larger instances can be easier than smaller ones.

Lemma 5. *Given a set S of n planar points of structural entropy $\mathcal{H}^*(S)$ and upper hull of size h, Kirkpatrick and Seidel's algorithm [23] is computing the upper hull of S in time $\mathcal{O}(n(1+\mathcal{H}^*(S))) \subseteq \mathcal{O}(n(1+\lg h)) \subseteq \mathcal{O}(n \lg n)$.*

Proof. (of Lemma 5) Consider a set S of n planar points of structural entropy of upper hull of size h and of $\mathcal{H}^*(S) = \sum_{i\in[h]} \frac{n_i}{n} \lg \frac{n}{n_i}$. We show that the complexity of the algorithm given by Kirkpatrick and Seidel [23] is $\mathcal{O}(n(1+\mathcal{H}^*(S)))$.

Kirkpatrick and Seidel's algorithm basically finds the points `min` and `max` of minimum and maximum abscissa in linear time, and selects in additional linear time in S' the points above or on the line $(\texttt{min}, \texttt{max})$, to finally call a recursive function $\texttt{Connect}(\texttt{min}, \texttt{max}, S')$ with those values, which returns the upper hull of S' and hence of S. The output sensitive result is essentially based on the analysis of the complexity of the recursive function $\texttt{Connect}(\texttt{min}, \texttt{max}, S)$, which computes the upper hull of S from `min` to `max`; and of the function $\texttt{BRIDGE}(S, m)$ which computes in linear time the edge of the upper hull crossing the vertical line passing by m. We refer the reader to Kirkpatrick and Seidel's definition and analysis of the `BRIDGE` function, and refine here the analysis of the `Connect` function.

The $\texttt{Connect}(\texttt{min}, \texttt{max}, S)$ function roughly goes as follows, when T is non empty. First it finds the median m of the points in S sorted by their abscissa (this can be done in linear time using the algorithm from Blum *et al.* [6]). Then it finds the (unique) edge (l, r) of the upper hull which crosses the vertical line passing by m (a simple linear program finds it in linear time). Then it partitions S into the set L of points to the left of l, and the set R of points to the right of r, discarding all the points of abscissa between those two points (which takes obviously linear time). Finally it calls recursively $\texttt{Connect}(\texttt{min}, l, L)$ and $\texttt{Connect}(r, \texttt{max}, R)$. Since the non recursive operations typically take time $c|S|$ linear in the size of S for some positive constant $c > 0$, Kirkpatrick and Seidel focused on the recursive function $f(n, h)$ and showed that it was $\mathcal{O}(n \lg h)$, yielding a complexity of $\mathcal{O}(n(1 + \lg h))$ when taking into account the linear time preprocessing. Instead, we study the how fast an edge of the upper hull is discovered in function of the number of points it "covers".

The intuition goes as follows: imagine that each edge (l, r) of the upper hull is a "bucket" such that, when m is in this bucket, (l, r) is discovered. The more points in the bucket corresponding to (l, r), the faster it will be discovered by the algorithm, since the number of points considered by each recursive call is divided by at least two at each new recursion. At the i-th recursive level the algorithm has found the 2^i buckets of size larger than $n/2^i$, and has eliminated all the corresponding points. In other words, the bucket i survives at most $\lg(n_i/n)$ recursive phases, costing cn_i operations in each of those phases, accounting for a total of $cn_i \lg(n_i/n)$ operations. Summing over the h buckets yield a complexity of $c \sum_{i\in[h]} n_i \lg(n_i/n)$, which, expressed in function of the structural entropy, yields the final expression of $cn \times \sum_{i\in[h]} \frac{n_i}{n} \lg(\frac{n_i}{n}) \in \mathcal{O}(n\mathcal{H}^*)$, yielding a complexity of $\mathcal{O}(n(1+\mathcal{H}^*))$ when taking into account the linear time preprocessing. □

One way to encode a certificate of the planar convex hull computed by the algorithm described in Lemma 5 would be to reorder the points (or equivalently to code a permutation over $[n]$, compressed using one of Barbay and Navarro's compression schemes [5]) so that the points in the shadow of an edge are immediately succeeding the points forming this edge: this might yield the most succinct encoding of the certificate, but depends on the input order, on which we did not make any assumption in this section. We describe another way, independent of the input order and taking advantage of the structural entropy.

Corollary 3. *A succinct representation of the certificate of the output of the algorithm described in Lemma 5 can be encoded in $n(2+\mathcal{H}^*(S))$ bits.*

Proof. (of Corollary 3) The idea is to assign to each edge of the upper hull a code of length decreasing with the number of points it covers, and to code a string associating to each point the edge covering it. One could use a Huffman tree [19] based on the normalized repartition of the points to generate an encoding of this string using $n\mathcal{H}^*(S)$ bits, but that would require to encode the dictionary associating the Huffman codes to each edge in $h \lg h$ bits. Instead, we simply use the Hu-Tucker algorithm [18] to generate codes ordered in the same order than `Rep`, i.e. in the order of the edges of the upper hull, which yield an encoding of the string using $n(2+\mathcal{H}^*(S))$ bits. □

Performing a similar analysis in three dimension, based on the algorithm from Edelsbrunner and Shi [13], yields an adaptive complexity of $\mathcal{O}(n(1+\mathcal{H}^*(S))\lg h)$, which is never better than the complexity $\mathcal{O}(n(1+\lg h))$ of Chan's algorithm [8]. There is no much hope to improve this result without major changes in the algorithm from Edelsbrunner and Shi [13], which is not optimal even over instances of fixed n and h. Their algorithm runs the two dimensional algorithm from Kirkpatrick and Seidel on the projection of the points on two vertical planes. Two points, originally in the shadow of the same single face of the upper hull in three dimension, can then be projected into the shadow of different edges of the planar convex hull on this plane. This implies that the structural entropy of the planar instances generated is not correlated to the structural entropy of the original instance in three dimensions.

3.3 Reductions and Lower Bounds

In this section we use the notion of reduction between pairs formed by a problem and a parameter of the analysis, a notion borrowed from parameterized complexity [16]. The main idea is to make sure that the reduction from (A, M_A) to (B, M_B) maps M_A-easy instances from a problem A to M_B easy instances from the problem B.

We first prove that the adaptive analysis of sorting multisets in function of their entropy, is reducible to the adaptive analysis of the computation of the convex hull in the plane in function of the structural entropy. In particular, it means that an adaptive algorithm for the convex hull over planar instances of fixed structural entropy yields an adaptive algorithm for sorting multisets over instances of fixed entropy, and that an adaptive lower bound on the computational complexity of sorting multisets of fixed entropy yields an adaptive lower for the computation of the convex hull over instances of fixed structural entropy.

Lemma 6. *Any algorithm computing the convex hull of an instance S of n points in the plane with structural entropy $\mathcal{H}^*(S)$ in time $f(n, \mathcal{H}^*(S))$ can be used to sort a multiset of n values and entropy $\mathcal{H}$ in time $\mathcal{O}(n) + f(n, \mathcal{H})$.*

Proof. (of Lemma 6) The reduction is quite simple: given a multiset of n values and entropy $\mathcal{H}$, map all the values to a set S of planar points via a function $\phi(x) = (x, x^2)$, as usual in reductions from sorting to the planar convex hull. Obviously this step takes linear time. The entropy $\mathcal{H}$ of the original multiset is by definition exactly the structural entropy $\mathcal{H}^*(S)$ of S, so computing the convex hull of S is done in time $f(n, \mathcal{H})$ which yields the final complexity of $\mathcal{O}(n) + f(n, \mathcal{H})$. □

We then prove an adaptive computational complexity lower bound for sorting multisets of fixed entropy, which shows that our analysis in two dimensions is tight over all deterministic and randomized algorithms in the algebraic decision tree model.

Lemma 7. *Given a positive integer n, a positive real number $\mathcal{H}$, and an algorithm A in the algebraic decision tree model for the computation of the convex hull in the plane; there is one instance of input size at most n and structural entropy at most $\mathcal{H}$ such that A performs $\Omega(n(1+\mathcal{H}))$ operations to compute its convex hull.*

Proof. (of Lemma 7) This is a simple reduction from the sorting of multisets, which lower bound can in turn be reduced to the encoding of multisets, itself obtained through a simple application of the definition of the entropy. $\square$

Combining the main results of this section yields the following Theorem:

Theorem 2. *Given a set S of n planar points of structural entropy $\mathcal{H}^*(S)$ and convex hull of size h, there is an algorithm computing the planar convex hull of S in time $\mathcal{O}(n(1+\mathcal{H}^*(S))) \subseteq \mathcal{O}(n(1+\lg h)) \subseteq \mathcal{O}(n \lg n)$. This is asymptotically optimal over all such instances, among all randomized or deterministic algorithms in the algebraic decision tree model.*

Of course, since one can sort values using an algorithm for the planar convex hull, this yields an entropy adaptive result for the sorting of multisets as well:

Corollary 4. *Given a multiset S of n values taken from $[h] = \{1, \dots, h\}$, there is an algorithm sorting S and counting the number of occurrences of each value in it in time $\mathcal{O}(n(1+\mathcal{H}^*(S))) \subseteq \mathcal{O}(n(1+\lg h))$, where $\mathcal{H}(S) = \sum_{i\in[h]} \frac{n_i}{n} \lg \frac{n}{n_i}$ and $\forall i \in [h]$ $n_i = \#\{x \in S \text{ s.t. } x = i\}$, and this is optimal in the comparison model.*

Not that although we prove the corollary above through a reduction, it is also easily proven through the adaptive analysis of algorithms such as a deterministic Quick Sort (which is the exact projection of Kirkpatrick and Seidel's convex hull algorithm), Merge Sort or an Insertion Sort using Splay trees.

4 Comparisons between Analysis

In this section we use the notion of reduction between measures of difficulty, which was introduced by Mannila [29] in order to compare measures of difficulty for a fixed problem.

Theorem 3. *Any algorithm (optimally) adaptive to the structural entropy $\mathcal{H}^*$ is (optimally) adaptive to the output size h.*

Proof. (of Theorem 3) The fact that any algorithm adaptive to the structural entropy is adaptive to the output size is trivial given that $\mathcal{H} \leq \lg h$. The fact that any algorithm optimally adaptive to the structural entropy is optimally adaptive to the output size comes from the lower bound from Theorem 7 $\square$

A direct consequence of this reduction is that the algorithms which are not adaptive to the output size (i.e. performing $\omega(n \lg h)$ operations for at least one instance of input size n and output size h) are not adaptive to the structural entropy. Hence it is easy to see that the naive implementations of quick-hull, divide and conquer, and other traditional algorithms are not adaptive to the structural entropy. This has an important consequence over the performance of Chan's output sensitive algorithm for the convex hull, which uses internally some traditional algorithms, albeit on small groups of points.

Corollary 5. *A straightforward implementation of Chan's algorithm is not adaptive to the structural entropy.*

Proof. (of Corollary 5) Chan's algorithm [8] is based on a linear search on the value of $\lceil \lg \lceil \lg h \rceil \rceil$, where each test checks if the size of the convex hull is smaller than a parameter m in $\mathcal{O}(n \lg m)$ orientation tests. To achieve this complexity, each test computes the hull of n/m groups of m points in $\mathcal{O}(n/m \times m \lg m) = \mathcal{O}(n \lg m)$ orientation tests, and tries to merge them all in a hull of m points using a variant of Jarvis's walk, where each of the m edges is obtained through n/m binary searches of $\lg m$ orientation tests, summing once again to $\mathcal{O}(n/m \times m \lg m) = \mathcal{O}(n \lg m)$.

Without loss of generality, assume that both the input size n and output size $m = h$ are given to the algorithm, which has only to compute the output in two phases, one computing the convex hull of n/h groups of h points and then cover them with h edges. Assume that h is a constant fraction of n, so that many points are not in the output. We give a specific instance formed by n points and of output size h, such that if those points are given in some specific order the algorithm already performs $\omega(n\mathcal{H})$ operations on this instance in the first phase, where $\mathcal{H}$ is the structural entropy of the instance, by design much lower than $\lg h$.

Consider an instance which convex hull is of size h and where all the points which are not in the convex hull are covered by exactly two successive edges, the same for all points. By definition, the structural entropy of the instance is $\mathcal{H} = \frac{n-h+1}{n} \lg \frac{n-h+1}{n} + \frac{h-1}{n} \lg \frac{1}{n} \in \mathcal{O}(1)$, corresponding to a repartition of $n - h + 1$ points for one pair of edges and exactly one for each other. On the other hand, the computation of the convex hull of the n/h groups of h points each takes $\Theta(n \lg h)$ orientation tests, whether the algorithm used recursively is adaptive to the output size or to the structural entropy: the groups of h points can have arbitrary structural entropy and output size.

Hence Chan's algorithm [8] performs $\Theta(n \lg h)$ operations (in fact, even $\Theta(n \lg n)$) on a particular instance where any algorithm optimally adaptive to the structural entropy would perform only $\mathcal{O}(n)$ operations. □

Generalizing Chan's algorithm so that it takes into account the structural entropy of planar instances might be the key to achieve an algorithm computing the three dimensional convex hull adaptively to the structural entropy in three dimensions, but it is not clear how to do so. Alternatively, Peyman Afshani has suggested that a different technique based on random sampling might yield adaptivity to the structural entropy in two and three dimensions, and potentially in higher dimensions.

5 Discussions

5.1 Adaptive Analysis

Whereas our techniques are mostly computational, generic, and lacking in geometrical intuition, our results are geometric in essence and answer many questions asked by computational geometrists, in personal communication (Olivier Devillers, Luca Castelli-Aleardi, Hee-Kap Ahn, Yoshio Okamoto) or in publications (Levcopoulos *et al.* [26]).

It is important not to deduce from this that adaptive analysis beyond output sensitivity inherently lacks geometric intuition: for each problem where there is a large gap between the worst and best case computational complexities, one can search for and find an explanation of this gap, formalize it through a measure of difficulty in order to produce a more precise analysis. There is no doubt that many of those "explanations" will be of geometric nature: in this our work is only preliminary, aiming to be generic so that others can adapt our technique to their own problem.

The development of more precise analysis techniques is required in order to bridge the widening gap between theory in practice, and it can only be initiated by theoreticians. Certainly, not all adaptive measures of difficulty correspond to realistic conditions on the input: between two given difficulty measures, the preferences must go to the most practical one. But since currently a few or no other measure of difficulty than the input size are known for most problems, we must study any difficulty measure to first increase our understanding and later focus on more practical one.

5.2 Application to Other Problems

Our techniques concerning the adaptivity to the input order can be generalized to any problem for which there is an incremental algorithm with best case complexity $\mathcal{O}(n)$ (the worst case complexity does not really matter, in our example it was $\mathcal{O}(n^2)$), and a linear time merging algorithm. A good inspiration for such a work on computational geometry would be the similar effort to generate "generic merge-sort algorithms" from Estivil-Castro and Wood [14].

Through our definition of adaptive reductions, our adaptive results on both the input order and the structural entropy can be applied in different settings provided through adequate mappings. Hopefully such reductions will yield to the definition of useful classes of equivalences, such that any improvement on some given computational problem can be propagated to similar problems without further work.

5.3 Open Problems

We distinguish three types of open problems: those related to the search of other adaptive measures of difficulty, those relative to the generalization of the marriage-before-conquest paradigm, and those relative to the generalization of the problems themselves, to higher dimension.

Inspiration from Adaptive Sorting. The reduction of sorting (multisets or permutation) to the computation of the convex hull in two dimensions (or to other problems above in the network of reductions) implies some "projection" of adaptive analysis from convex hull to sorting. Such "projected" adaptive analysis can be trivial: the output sensitivity analysis of the convex hull does not yield any improvement on the analysis of sorting permutations (although it does in a minor way for the analysis of sorting multisets). But there are *many* adaptive analysis and algorithms for sorting permutations. It seems likely that those analysis have equivalents in more complex problems such as the convex hull, of which they are only projections.

Combining (elegantly) distinct analysis. It is trivial to combine two algorithms adaptively optimal for distinct measures of difficulty into a single asymptotically optimally adaptive algorithm to both measures: one just needs to run both algorithms (each optimal for its own adaptive measure of difficulty) in parallel, and to stop the computation as soon as one of them terminates. In the case of the computation of the planar convex hull, a more elegant way can be achieved by running the linear time partitioning algorithm described in Lemma 1 to produce ρ solved sub-instances, and then use Nielsen and Yvinec's [33] algorithm to compute the convex hull of those convex objects. In the case of the computation of the convex hull in three dimensions, a more elegant way can be achieved by combining a variant of the partitioning of Lemma 1, where the longest branches of the merging scheme are rebalanced with neglectable impact on the performance following a technique initiated by Buro [7]; with a variant of Chan's algorithm, rewritten as a merging tree, to yield an algorithm running in time $\mathcal{O}(n(1+\min\{\lg h, \mathcal{H}(\texttt{Runs})\}))$.

Marriage before Conquest. The paradigms of "marriage before conquest" introduced by Kirkpatrick and Seidel [23] and of "merging and spanning" introduced by Chan [8] both yield an output sensitive $\mathcal{O}(n \lg h)$ complexity for the computation of the convex hull in two dimensions. The known algorithms implementing those two paradigms differ in two aspects:

- Kirkpatrick and Seidel's algorithm yields a better adaptive complexity for the computation of the convex hull in two dimensions, whereas Chan's algorithm can be easily generalized to compute the convex hull in three dimensions.
- The performance of Kirkpatrick and Seidel's algorithm is rotation dependant, for both the output sensitive analysis and the structural entropy analysis, whereas the performance of Chan's algorithm is rotation independent.

It does not seem impossible to generalize the "marriage before conquest" from Kirkpatrick and Seidel to compute the convex hull in three dimensions. The main difficulty resides in the fact that each "bridge" consists in a sequence of faces, which lengths should be taken into account in the difficulty of the instance, which will exacerbate even further the rotation dependency of the complexity of the algorithm

The real challenge consists in finding an algorithm which would be adaptive to a rotation independent definition of the structural entropy. One way to achieve this could be to combine the two paradigms into the same algorithm in a more elegant way than simulating both algorithms in parallel, but it does not seem trivial.

Generalization to Higher Dimensions. Another sizable challenge concerns similar problems in higher dimensions. The worst case complexity of known algorithms over instances of fixed size grows exponentially with the dimension of the space, along with the worst case size of the input. Since instances in lower dimensions are mere particular cases of the instances in higher dimension, the gap between the best and worst case complexities is also increasing exponentially with the dimension. This should create many opportunities for adaptive analysis, to separate "easy" instances from difficult ones in a continuous way. An example would be to consider instances of high dimensionality formed from the union of a few sub-instances of lower dimensionality can be solved faster, reducing the difficulty for such instances to the search for their decomposition in simpler instances.

Acknowledgements:

This paper would probably never have been started without the encouragements from Luca Castelli Aleardi, Olivier Devillers, Jack Snoeyink, Therese Biedl and many others from the community of Computational Geometry. It would never have been terminated either without the productive comments from Peyman Afshani and Luca Castelli Aleardi. Thanks to all of them.

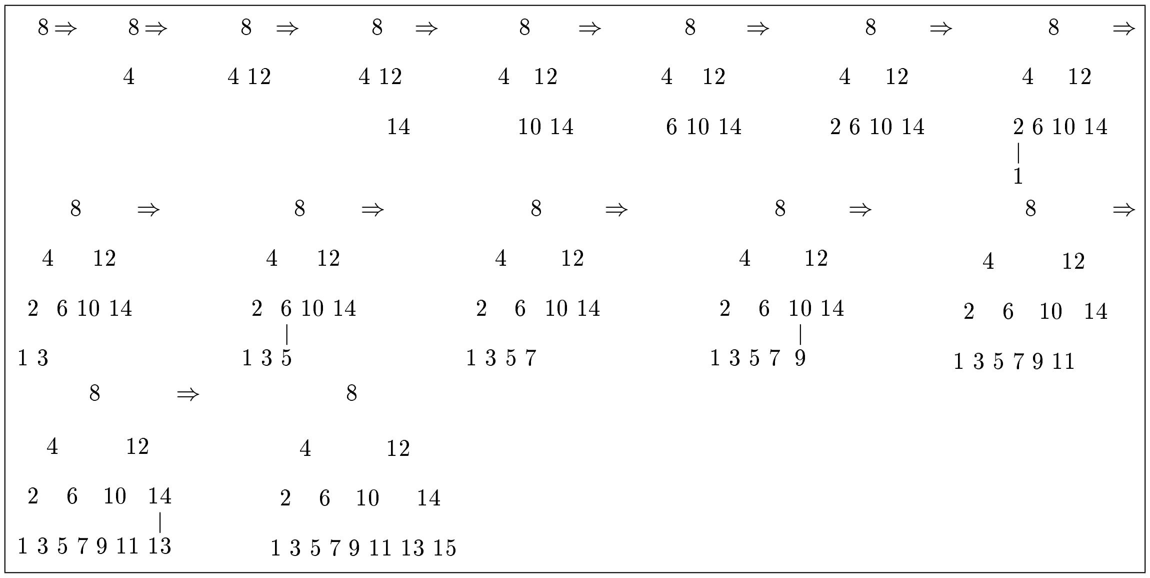

Fig. 1. A 3-progressive sequences for sorting: Consider the sequence $(8, 4, 12, 14, 10, 6, 2, 1, 3, 5, 7, 9, 11, 13, 15)$. The insertion of those values in a binary search tree yields a perfectly balanced tree, without the need of any rebalancing operation. Supporting the parent operator and maintaining a pointer to the last node created, at most 3 comparisons are required for each new values, by simple application of Snoeyink and Kreveld [37]'s result.

Bibliography

[1] P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In *Advances in Discrete and Computational Geometry*, pages 1–56. American Mathematical Society, 1999.

[2] A. Aggarwal, L. Guibas, J. Saxe, and P. Shor. A linear time algorithm for computing the voronoi diagram of a convex polygon. In *STOC '87: Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 39–45, New York, NY, USA, 1987. ACM.

[3] J. Barbay and E. Chen. Adaptive planar convex hull algorithm for a set of convex hulls. In *Proceedings of the 20th Annual Canadian Conference on Computational Geometry, CCCG 2008, August 20-22, 2008, Montreal, Canada*, 2008.

[4] J. Barbay and C. Kenyon. Alternation and redundancy analysis of the intersection problem. *ACM Trans. Algorithms*, 4(1):1–18, 2008.

[5] J. Barbay and G. Navarro. Compressed representations of permutations, and applications. In *Proceedings of STACS, to appear*, 2009.

[6] M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Tarjan. Linear time bounds for median computations. In *STOC '72: Proceedings of the fourth annual ACM symposium on Theory of computing*, pages 119–124, New York, NY, USA, 1972. ACM.

[7] M. Buro. On the maximum length of huffman codes. *Information Processing Letters*, 45:219–223, 1993.

[8] T. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *GEOMETRY: Discrete & Computational Geometry*, 16, 1996.

[9] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(5):485–524, 1991.

[10] B. Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM J. Comput.*, 21(4):671–696, 1992.

[11] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Adaptive set intersections, unions, and differences. In *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 743–752, 2000.

[12] H. Djidjev and A. Lingas. On computing the Voronoi diagram for restricted planar figures. In *Proc. 2nd Worksh. Algorithms and Data Structures*, pages 54–64. Springer-Verlag, LNCS 519, 1991.

[13] H. Edelsbrunner and W. Shi. An o(n log2h) time algorithm for the three-dimensional convex hull problem. *SIAM J. Comput.*, 20(2):259–269, 1990.

[14] V. Estivill-Castro and D. Wood. A survey of adaptive sorting algorithms. *ACM Computing Surveys*, 24(4):441–476, 1992.

[15] P. Ferragina, G. Manzini, V. Mäkinen, and G. Navarro. Compressed representations of sequences and full-text indexes. *ACM Transactions on Algorithms (TALG)*, 3(2):article 20, 2007.

[16] J. Flum and M. Grohe. *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[17] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams. *ACM Trans. Graph.*, 4(2):74–123, 1985.

[18] T. Hu and A. Tucker. Optimal computer-search trees and variable-length alphabetic codes. *SIAM Journal of Applied Mathematics*, 21:514–532, 1971.

[19] D. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the I.R.E.*, 40(9):1090–1101, 1952.

[20] J. Jansson, K. Sadakane, and W.-K. Sung. Ultra-succinct representation of ordered trees. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 575–584. ACM, 2007.

[21] R. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters*, 2:18–21, 1973.

[22] D. G. Kirkpatrick. Efficient computation of continuous skeletons. In *SFCS '79: Proceedings of the 20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 18–27, Washington, DC, USA, 1979. IEEE Computer Society.

[23] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM J. Comput.*, 1986. 15(1):287–299.

[24] R. Klein and A. Lingas. A linear-time randomized algorithm for the bounded voronoi diagram of a simple polygon. In *SCG '93: Proceedings of the ninth annual symposium on Computational geometry*, pages 124–132, New York, NY, USA, 1993. ACM.

[25] D. E. Knuth. *Art of Computer Programming, Volume 3: Sorting and Searching (2nd Edition)*. Addison-Wesley Professional, April 1998.

[26] C. Levcopoulos, A. Lingas, and J. S. B. Mitchell. Adaptive algorithms for constructing convex hulls and triangulations of polygonal chains. In *SWAT '02: Proceedings of the 8th Scandinavian Workshop on Algorithm Theory*, pages 80–89, London, UK, 2002. Springer-Verlag.

[27] C. Levcopoulos and O. Petersson. Sorting shuffled monotone sequences. *Inf. Comput.*, 112(1):37–50, 1994.

[28] C. S. M. Denny. Encoding a triangulation as a permutation of its point set. In *9th Canadian Conference on Computational Geometry*, 1997.

[29] H. Mannila. Measures of presortedness and optimal sorting algorithms. In *IEEE Trans. Comput.*, volume 34, pages 318–325, 1985.

[30] D. Marx. Parameterized complexity of constraint satisfaction problems. In *Proceedings of 19th Annual IEEE Conference on Computational Complexity*, pages 139–149, 2004.

[31] K. Mehlhorn and S. Näher. Leda: a platform for combinatorial and geometric computing. *Commun. ACM*, 38(1):96–102, 1995.

[32] E. Moura, G. Navarro, N. Ziviani, and R. Baeza-Yates. Fast and flexible word searching on compressed text. *ACM Transactions on Information Systems (TOIS)*, 18(2):113–139, 2000.

[33] F. Nielsen and M. Yvinec. Output-sensitive convex hull algorithms of planar convex objects. *Internat. J. Comput. Geom. Appl*, 8:39–65, 1995.

[34] F. P. Preparata and S. J. Hong. Convex hulls of finite sets of points in two and three dimensions. *Commun. ACM*, 20:87–93, 1977.

[35] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

[36] R. Seidel. A method for proving lower bounds for certain geometric problems. Technical report, Cornell, Ithaca, NY, USA, 1984.

[37] J. Snoeyink and M. van Kreveld. Good orders for incremental (re)construction. In *SCG '97: Proceedings of the thirteenth annual symposium on Computational geometry*, pages 400–402, New York, NY, USA, 1997. ACM.