Computational Geometry: Theory and Applications 2 (1993) 267–278
Elsevier

COMGEO 135

Constructing the convex hull of a partially sorted set of points

Michael T. Goodrich*

Department of Computer Science, The Johns Hopkins University, Baltimore, MD 21218-2694, USA

Communicated by Raimund Seidel
Accepted 19 February 1992

Abstract

Goodrich, M.T., Constructing the convex hull of a partially sorted set of points, Computational Geometry: Theory and Applications 2 (1993) 267–278.

In this paper we give an optimal algorithm for constructing the convex hull of a partially sorted set S of n points in $\mathbb{R}^2$. Specifically, we assume S is represented as the union of a collection of non-empty subsets $S_0, S_1, S_2, \ldots, S_m$, where the x-coordinate of each point in S_i is smaller than the x-coordinate of any point in S_j if $i<j$. Our method runs in $O(n \log h_{max})$ time, where h_{max} is the maximum number of hull edges incident on the points of any single subset S_i. In fact, if one is only interested in finding the hull edges that 'bridge' different subsets, then our method runs in $O(n)$ time.

1. Introduction

Suppose we are given a set S of n points in the plane. We are interested in constructing the convex hull of S, which we denote by CH(S), that is, the smallest convex set containing the points in S. This problem is perhaps the most-studied problem in computational geometry, and has a host of applications (see [8, 14, 17]). Typically, the problem of constructing CH(S) is divided into that of constructing the upper hull, UH(S), and lower hull, LH(S), of S, where UH(S) (respectively, LH(S)) is defined to be the edges on the boundary of CH(S) that are visible from above (respectively, below). That is, any vertical line L intersects the boundary of CH(S) in a point p on UH(S) and a point q on LH(S) with either $p=q$ or p being directly above q. (See Fig. 1.) Thus, it is

Correspondence to: Michael T. Goodrich, Department of Computer Science, The Johns Hopkins University, Baltimore, MD 21218-2686, USA.
* This research was supported by the National Science Foundation under Grant CCR-9003299.

0925-7721/93/$06.00 © 1993 — Elsevier Science Publishers B.V. All rights reserved

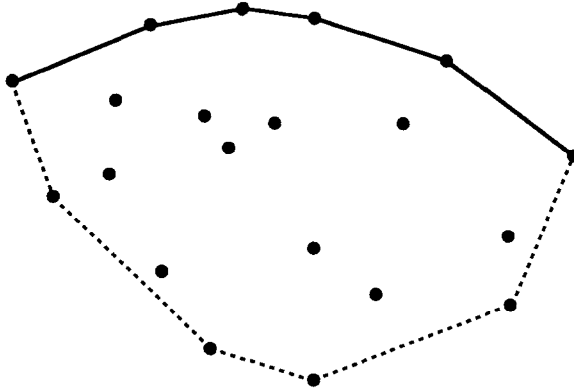


Fig. 1. A convex hull. UH(S) is shown with solid edges, and LH(S) is shown with dashed edges.

sufficient for one to show how to construct UH(S), as the method for constructing LH(S) is similar, and merging UH(S) and LH(S) into CH(S) is quite easy.

1.1. Review of previous results

Before we give our algorithm for constructing UH(S), let us first review a few results that relate to convex hull construction. We begin with a lemma that shows that one can 'probe' UH(S) in linear time.

Lemma 1.1. *Given a vertical line L, and an m-point subset S' of S, one can determine the edge e of* UH(S') *that intersects L (or that no such edge exists) in* O(m) *time.*

Proof. The proof follows immediately, by duality [8, 9, 13], from the methods of Dyer [7] and Megiddo [15, 16] for solving a fixed-dimensional linear program in linear time. □

Typically, the edge e is referred to as the *bridge* for S' with respect to L [13]. Kirkpatrick and Seidel [13] use this lemma as a building block for an asymptotically fast convex hull algorithm for the case when S is unsorted, proving the following lemma.

Lemma 1.2 (Kirkpatrick and Seidel [13]). *Given a set S of n points in the plane, one can construct* UH(S) *in time* $O(n \log h)$, *where* $h = |\mathrm{UH}(S)|$.

The method of Kirkpatrick and Seidel for proving this lemma involves a paradigm they call *marriage-before-conquest*, where one divides a set of points into two equal halves by a vertical line L, uses Lemma 1.1 to find the bridge e with respect to L, and then recurses on the points of each half that are not

directly below e. This results in an algorithm whose running time is optimal, since $\Omega(n \log h)$ is a lower bound for the running time of constructing UH(S) [13, 19]. Nevertheless, if all the points of S are given in sorted order, say by increasing x-coordinates, then, by the following lemma, one can do better than this.

Lemma 1.3. *Given a set S of n points in the plane sorted by increasing x-coordinates, one can construct* UH(S) *in time* O(n).

Proof. The method follows by a straightforward adaptation of the convex hull algorithm due to Graham [11] (indeed, the proof of this lemma is often given as a homework exercise in computational geometry courses). In particular, the approach for proving this lemma, commonly referred to as the *Graham scan* method [17], is to construct the upper hull on-line by considering the points one at a time by increasing x-coordinates. It is clearly optimal for any sorted point set S. □

In addition to the methods above, there are a large number of other interesting convex hull algorithms (e.g., see [1, 2, 3, 4, 6, 8, 10, 12, 14, 17, 18]), but none are more efficient than the two above for their respective versions of the problem.

1.2. Our results

In this paper we address an intermediate version of the convex hull problem, one in which the input is *partially* sorted. In particular, we assume we are given a set S of n points in the plane, and a set R of m vertical lines that induce a partitioning Π of S into subsets $S_0, S_1, S_2, \ldots, S_m$ in the natural way, i.e., the x-coordinate of each point in S_i is smaller than the x-coordinate of any point in S_j if $i < j$, and S_i is separated from S_{i+1} by a line in R, for $i = 1, 2, \ldots, m$. Without loss of generality, we assume that $m < n$. If this is not the case, then we can 'collapse' each pair of lines that have no points of S between them—these lines will intersect the same edge of UH(S).

Our method for constructing UH(S), which we describe in the subsequent sections, can be viewed as a nontrivial 'blend' of the Graham scan and marriage-before-conquest methods. In particular, we show how to find the edges of UH(S) that intersect the vertical lines in R in O(n) time. We refer to these edges as the *bridge edges* of S and R, and the problem of finding them as the *multiple bridge finding* problem. Given a solution to the multiple bridge finding problem, it is a simple matter to then apply Lemma 1.2 on the points of each S_i not covered by a bridge edge to derive a running time that is O($n \log h_{\max}$), where $h_{\max}$ is the maximum, taken over the subsets S_i, of the number of edges in UH(S) that are incident on points in S_i. Thus, let us restrict our attention to the multiple bridge finding problem. In the following section we describe a simple, but inefficient method, and we then show how to use it as a stepping stone to a linear-time method.

2. A simple stack algorithm

The algorithm we describe in this subsection solves the multiple bridge finding problem in a fashion very reminiscent of Graham's convex hull algorithm [11]. The output of our algorithm is a chain of edges that describes a solution to the multiple bridge finding problem. To be precise, if $E = (e_1, e_2, \ldots, e_h)$ is a list of the bridge edges for S with respect to R, listed from left to right, then we define the *partition hull* of S with respect to R as the chain that is formed by first removing any duplicate edges from E and then inserting a (possibly degenerate) edge joining the right endpoint of e_i with the left endpoint of e_{i+1}, for each $i = 1, 2, \ldots, h-1$. This forms a chain $H = (e_1, f_2, e_2, f_3, \ldots, f_{h'}, e_{h'})$, and this is what our method returns. We refer to each f_i as a *pseudo-edge*. (See Fig. 2.) For any point p we use $x(p)$ and $y(p)$ to respectively denote the s- and y-coordinate of p, and we assume that any non-vertical edge $e = \overline{uv}$ is represented so that $x(u) < x(v)$.

Since we are confining our attention to the edges of UH(S) that intersect R, we use the term *upper-convex* to refer to a chain C that is monotone with respect to the x-axis and such that each consecutive pair of edges in C makes a *right turn* when the edges are transversed from left to right. That is, if $d = \overline{uv}$ and $f = \overline{vw}$, then w is either on or to the right of the oriented line determined by e. (Note that this definition allows for collinear edges on an upper-convex chain.) The following lemma establishes an important property of H.

Lemma 2.1. *H is an upper-convex chain.*

Proof. If a consecutive pair of edges (e_i, f_j) or (f_j, e_{j+1}) in H form a left turn, then either e_j or e_{j+1} cannot be an edge of UH(S). The proof follows, then, since each e_j is an edge of UH(S) by definition. □

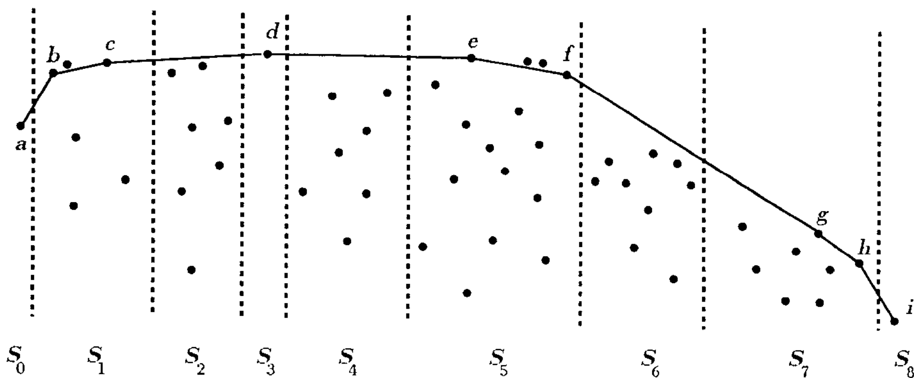


Fig. 2. An example partition hull. The edges (a, b), (c, d), (f, g), and (h, i) are bridges; the edges (b, c), (d, d), (e, f) and (g, h) are pseudo-edges (with (d, d) being a degenerate pseudo-edge).

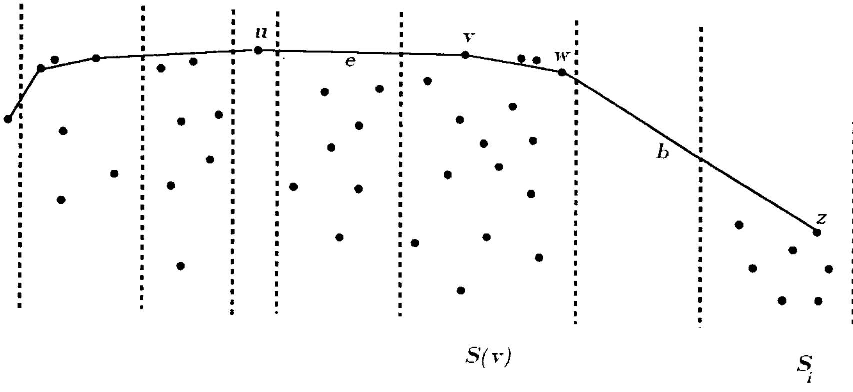
Given a partition hull H for S with respect to R it is a simple matter to reconstruct a solution to the multiple bridge finding problem by removing the pseudo-edges from H and then merging this list of bridge edges with the list R (to enumerate all the duplicate bridge edges for consecutive lines in R). Thus, without loss of generality, we may concentrate on the problem of computing the partition hull of S with respect to R. We consider each subset S_i in turn, while maintaining the partition hull of the subsets considered so far. We store the vertices of this partition hull on a stack σ. With each vertex v, we store a pointer to the subset, S_i that contains v, and we let $S(v)$ denote this subset. We initially push the leftmost vertex p of S_1 on σ (a vertex that must be on UH(S)). This will cause our partition to always begin with a pseudo-edge f_1 whose left endpoint is p, a convention that will simplify the comparison operation we use in our method.

Let S_i be the next subset to be considered, let v denote the vertex at the top of the stack σ, and let u denote the vertex below v in σ (if u exists). Note that the edge $d = \overline{uv}$ is a bridge for the subsets considered so far. We use the bridge-finding procedure (Lemma 1.1) to find the bridge $b = \overline{wz}$ between $S(v)$ and S_i. Let w be the left endpoint of b (a point in $S(v)$). We say that e and b form a *right turn*, if $uvwx$ is a upper-convex chain (possibly with $v = w$, so that $\overline{vw}$ is an degenerate pseudo-edge) or if u does not exist (in which case $S(v) = S_1$). Otherwise, we say e and b form a *left turn*. (See Fig. 3.) If e and b from a right turn, then push w and z, and this completes this iteration (for S_i). Otherwise, we pop v and u, and repeat this test. The algorithm terminates when we complete the iteration for S_m.

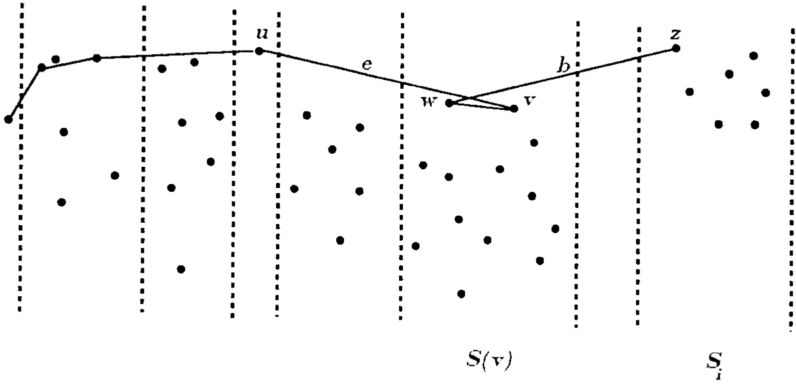
By a simple induction argument, the above algorithm, which we call the *simple stack* method, produces a solution to the multiple bridge finding problem. We leave the details of this argument to the interested reader.

The running time of this method is $O(ms_{\max})$, where $s_{\max}$ is the size of the largest subset S_i. The proof of this fact is based on an accounting argument that is very similar to the accounting argument for Graham's convex hull algorithm [11, 17]. The main idea is that for each step in the simple stack algorithm we spend at most $O(s_{\max})$ time performing a comparison between an $S(v)$ and S_i, and we can charge the work for the operation to $S(v)$ if we must perform a pop or, alternatively, to S_i if we must perform a push. Each subset has its 'representative vertices' pushed or popped at most once; hence, there are $O(m)$ stack operations. Unfortunately, the product $ms_{\max}$ can be as bad as $\Omega(n^2)$. Moreover, an example that has this as its time bound can be constructed by having $|S_1| = n/2$ and $|S_i| = 1$ for $i \in \{2, 3, \ldots, n/2\}$ such that the above algorithm must perform a bridge computation between S_1 and each S_i in turn. (See Fig. 4.) All is not lost, however, for the above method does result in an efficient algorithm if all the subsets are, more or less, the same size, as the following lemma shows.

Lemma 2.2. *If there is an integer s such that $s/2 \leq |S_i| \leq 2s$, for all $i \in \{1, 2, \ldots, m\}$, then the simple stack method constructs the partition hull of S with respect to R in* $O(n)$ *time.*



(a)



(b)

Fig. 3. Illustrating an iteration in the simple stack algorithm. The edges e and b form a right turn in (a) and form a left turn in (b).

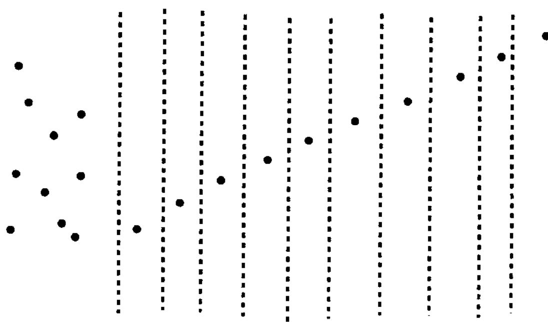


Fig. 4. A counter-example. The figure illustrates an example that forces the simple stack algorithm to take $\Omega(n^2)$ time.

Proof. In this case $m \leqslant 2n/s$ and $s_{\max} \leqslant 2s$, so $ms_{\max}$ is $O(n)$. □

We show below how to use the simple stack algorithm as a stepping stone to a method that runs in linear time for every problem instance.

3. Merging partition hulls with buckets

Before we give our optimal method, however, we first describe a method for merging two partition hulls H_1 and H_2 that are separated by a vertical line L (with H_1 to the left of H_2). Specifically, suppose we are given two sets of points S_1 and S_2 and two sets of vertical lines R_1 and R_2, such that S_1 and R_1 are separated from S_2 and R_2 by the vertical line L. In addition, assume that we have partition hulls H_1 H_2, such that H_1 is defined on S_1 with respect to R_1, and H_2 is defined on S_2 with respect to R_2. We show in this section how to efficiently produce the partition hull H defined on $S_1 \cup S_2$ with respect to $R_1 \cup \{L\} \cup R_2$.

We define the *weight* of a vertex v in a partition hull to be the size of $S(v)$, and use s_v to denote this value. Let $s = \max\{s_v : v \in H_2 \cup H_2\}$, i.e., s is the weight of the largest-weight node in $H_1 \cup H_2$. We form two 'buckets' B_l and B_r for H_1 and H_2, respectively, where B_l is the maximal suffix of nodes of H_1 such that sum of their weights is less than $2s$, and B_r is the maximal prefix of nodes of H_2 such that sum of their weights is less than $2s$. Note that the total weight of the nodes in B_l (respectively, B_r) is at least s, since no vertex has weight more than s.

We compute the bridge b between B_l and B_r, which can be done in $O(s)$ time, since with each partition hull vertex v we store a pointer to $S(v)$. (See Fig. 5.) We then pop from H_1 any nodes of B_l that b makes redundant, i.e., we perform the

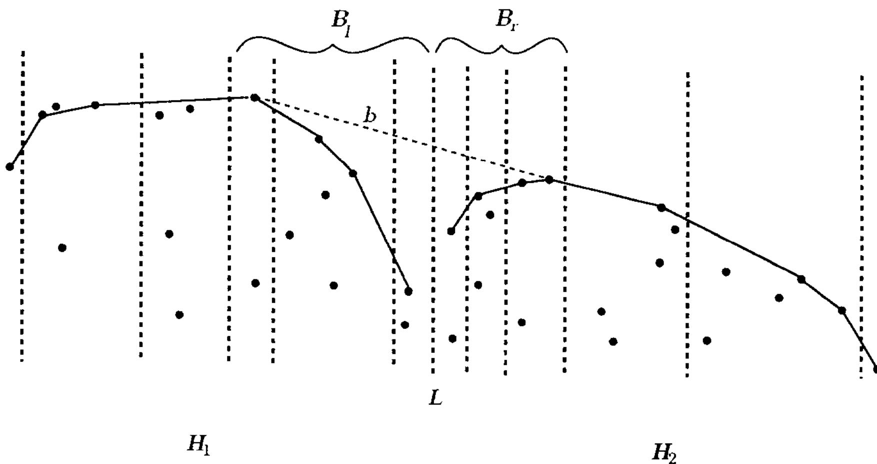


Fig. 5. Merging two partition hulls with buckets. In this case, $s = 5$.

repeated comparison test as in the simple stack operation. Note, however, that we never need to compute any new bridges while we are comparing two nodes in the bucket, B_l, since b is the bridge for all the points of B_l. If this popping action ultimately removes all the points from B_l, then we construct a new B_l from the remaining nodes in H_1 by again taking a maximal suffix whose total weight is less than $2s$. It is only at this point that we compute a new bridge b between B_l and B_r. We continue this bucketing and iteration on H_1 until the bridge b finally forms a right turn with the previous edge in H_1.

Once b forms a right turn with the previous edge in H_1, we then use b to perform an operation symmetric to the comparisons with the nodes of B_l to then pop from H_2 any edges of B_r that b makes redundant. If this popping action ultimately removes all the points from B_r, then we construct a new B_r from the remaining nodes in H_2, i.e., we take a maximal prefix whose total weight is less than $2s$. As in the previous case, it is only at this point that we compute a new bridge b between B_l and B_r. If we do compute a new bridge at this point, then we repeat our comparison operation with H_1 (not H_2), repeating the computation of the previous paragraph. We do not compare b with nodes of H_2 until we have once again terminated our comparisons with H_1. That is, the comparison operations with H_1 form the 'inner loop' of our computation. We terminate the 'outer loop' of our computation when we finally find a bridge b that forms a right turn with the previous edge in H_1 and the successive edge in H_2. The new partition hull H is formed by concatenating the remaining nodes of H_1, the endpoints of b, and the remaining nodes of H_2.

The correctness of this computation follows by an argument similar to that used in the simple stack algorithm, which, as in that case, we leave to the interested reader. Considering the time complexity, note that we must spend $O(s)$ time each time we compute a bridge. Note, however, that before we compute a new bridge, we must pop a collection of nodes with total weight at least s. Moreover, between bridge computations, all the comparison operations can be implemented in $O(s)$ time. Thus, we can charge all but the first and last bridge computations to the points in subsets that are never again to be considered (for they were 'popped'). In addition, if H_1 and H_2 are represented as doubly-linked lists, the final concatenation process can be implemented in $O(1)$ time. Therefore, we have the following lemma.

Lemma 3.1. *Given the partitional hulls H_1 and H_2, as described above, one can compute the partition hull H, defined on $S_1 \cup S_2$ with respect to $R_1 \cup \{L\} \cup R_2$, in $O(r+s)$ time, where r is the total weight of all the nodes removed during the process and s is the weight of the largest-weight node in $H_1 \cup H_2$.*

Having presented and analyzed our method for merging partition hulls, we are now ready to give our optimal method for constructing a partition hull.

4. A stack-of-stacks method

Suppose we are given a set S of n points in the plane, and a set R of m vertical lines that induce a partitioning Π of S into subsets $S_0, S_1, S_2, \ldots, S_m$ in the natural way. In this section we describe our method for constructing the partition hull H of S with respect to R in $O(n)$ time. Our method is based on the idea of maintaining a stack Γ of simple stacks, where each simple stack σ represents a partition hull restricted to subsets of S that are no larger than a weight value s_σ associated with σ. Moreover, we maintain the invariant that the weight associated with a stack σ in Γ is at most half that of the next deeper stack in Γ.

The details of our method are as follows. As an initialization step, we augment S with a new set S_{m+1}, where S_{m+1} is a set of n points 'at $-\infty$' that are below and to the right of all the points of S. That is, if we let q be a point in S_{m+1}, and $e = \overline{uv}$ be any nonvertical edge determined by points $u, v \in S$, and we define $f = \overline{vq}$, then (e, f) is a right turn. We initialize Γ to contain a single (empty) stack σ with weight $s_\sigma = 0$.

The Test for S_i. Let σ denote the topmost stack in Γ, and let S_i be the next subset to be considered. There are two cases:

Case 1. $|S_i| \geqslant s_\sigma/2$, *i.e., S_i is either appropriate or too large for σ.*

Let σ^* denote the stack just below σ in Γ. There are two subcases:

Subcase 1 (a): $|S_i| < s_{\sigma^*}/2$ (*i.e., S_i is too small for σ^**) *or σ^* does not exist.*

Then we use the partition hull merging method described in the previous section to merge the partition hull represented by σ with the (degenerate) partition hull represented by the rightmost point in S_i, giving σ_{new}. We then update $s_{\sigma_{\text{new}}}$ to be the smallest power of 2 greater than or equal to $\max(|S_i|, s_\sigma)$. This completes the processing for S_i.

Subcase 1 (b): $|S_i| \geqslant s_{\sigma^*}/2$, *i.e., S_i is either appropriate or too large for σ^*.*

Then we merge σ^* and σ using the partition hull merging procedure described in the previous subsection, with σ^* playing the role of H_1 and σ playing the role of H_2. We give the resulting partition hull σ_{new} weight $s_{\sigma_{\text{new}}} = s_{\sigma^*}$. We then pop σ and σ^* and push σ_{new} onto Γ (so that it will play the role of the topmost stack) and we repeat the test for S_i.

Case 2: $|S_i| < s_\sigma/2$, *i.e., S_i is too small for σ.*

Then we create a new stack σ_{new}, and give it as its weight, $s_{\sigma_{\text{new}}}$, the smallest power of 2 greater than or equal to $|S_i|$ (i.e., $2^{\lceil \log |S_i| \rceil}$). We push the leftmost vertex of S_i onto σ_{new} and then push σ_{new} onto Γ. This completes the processing for S_i.

We repeat the above iterative procedure until we exhaust all the subsets in the partition for S. Since we padded the partition of S with a set S_{m+1} of size n, Γ will contain a single stack σ at the end of the procedure, and σ will be a solution to the partition hull problem for S. The correctness of this method follows by a

simple induction argument based on the correctness of our method for merging partition hulls with buckets.

Let us, therefore, analyze the running time of this stack-of-stacks method for computing partition hulls. We provide for all the work used by our algorithm by a simple accounting scheme, showing that the entire method requires only $O(n)$ time. Let us take each case in turn. In Subcase 1(a) we require $O(r+|S_i|)$ time, where r is the total size of nodes removed during the partition hull merging procedure, by Lemma 3.1. Thus, we can account for the work in this case by charging each point in S_i one charge and also charging each point in a removed subset one charge. The accounting for Subcase 1(b) is a little more involved, however. So, let k be the number of times we must iterate Subcase 1(b) before we can perform Subcase 1(a), and let $\sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_k$ denote the sequence of stacks in Γ that we iteratively merge before performing Subcase 1(a), listed as they originally appeared in Γ (with σ_i being below σ_{i+1}). By Lemma 3.1, the total time needed to merge all these stacks is $O(r+s_{\sigma_1}+s_{\sigma_2}+s_{\sigma_3}+\cdots+s_{\sigma k})$, where r is the total size of all the nodes removed during the merges. Note, however that

$$\begin{aligned} s_{\sigma_1}+s_{\sigma_2}+s_{\sigma_3}+\cdots+s_{\sigma_k} &\leqslant s_{\sigma_1}+s_{\sigma_1}/2+s_{\sigma_1}/4+\cdots+s_{\sigma_1}/2^k \\ &\leqslant 2s_{\sigma_1}\leqslant 4|S_i|. \end{aligned}$$

Thus, we can account for the time we spend performing all the iterations of Subcase 1(b) by charging each point in S_i four charges and each point in a removed subset one charge. In Case 2 we require $O(|S_i|)$ time, which we can account for by charging each point in S_i one charge. Thus, each point in S gets charged at most $O(1)$ times during the entire procedure. This gives us the following theorem.

Theorem 4.1. *Given a set S of n points in the plane, partitioned in the natural way by a set R of m vertical lines, then one can construct the partition hull for S with respect to R in* $O(n)$ *time.*

This, in turn, gives us the following two corollaries.

Corollary 4.2. *Given S and R as in the theorem, one can solve the multiple bridge finding problem for S and R in* $O(n)$ *time.*

Corollary 4.3. *Given S and R as in the theorem, one can construct* $\mathrm{UH}(S)$ *in* $O(n\log h_{\max})$ *time, where $h_{\max}$ is the maximum number of edges of* $\mathrm{UH}(S)$ *incident upon a single subset of S induced by R.*

5. Conclusion

We have given an efficient method for constructing the convex hull of a partially sorted set of points in the plane. The definition of 'partially sorted' that

we have chosen to use is that of an induced partition of the set of points by a collection of vertical lines. As a direction for future research one could also imagine other definitions of 'partially sorted'.

Incidentally, our original interest for this work was actually motivated by the 3-dimensional convex hull algorithm of Edelsbrunner and Shi [9]. In their algorithm they repeatedly construct 2-dimensional convex hulls of the projections of the 3-dimensional points. If one uses the 2-dimensional convex hull to partially sort the set of points in $\mathbb{R}^3$, then this iterative process can be viewed as that of repeatedly constructing 2-dimensional convex hulls of partially sorted sets (which can then be further refined using the newly-constructed hull). Unfortunately, if applied to the implementation as stated in [9], this approach only eliminates half of the bottle-neck procedure calls in their algorithm; hence, can only improve their running time of $O(n \log^2 h)$ by a constant factor (recall that h is the size of the 3-D hull in this case). Interestingly, however, as recently shown by Chazelle and Matoušek [5], one can achieve an $O(n \log h)$ running time for constructing a 3-dimensional convex hull using a completely different method.

Acknowledgement

We would like to thank the anonymous referees for many helpful comments.

References

[1] A. Aggarwal, B. Chazelle, L. Guibas, C. Ó'Dúnlaing and C. Yap, Parallel computational geometry, Algorithmica 3 (1988) 293–328.

[2] M.J. Atallah and M.T. Goodrich, Efficient parallel solutions to some geometric problems, Parallel Distributed Comput. 3 (1986) 492–507.

[3] B. Chazelle, On the convex layers of a planar set, IEEE Trans. Inform. Theory 31 (1985) 509–517.

[4] B. Chazelle, An optimal convex hull algorithm for point sets in any fixed dimension, Tech. Report CS-TR-336-91, Dept. of Computer Science, Princeton University, 1991.

[5] B. Chazelle, private communication.

[6] K. Clarkson and P. Shor, Applications of random sampling in computational geometry, II Discrete Comput. Geom. 4 (1989) 387–421.

[7] M.E. Dyer, Linear time algorithms for two- and three-dimensional linear programs, SIAM J. Comput. 13 (1984) 31–45.

[8] H. Edelsbrunner, Algorithms in Combinatorial Geometry (Springer, Berlin, 1987).

[9] H. Edelsbrunner and W. Shi, An $O(n \log^2 h)$ time algorithm for the three-dimensional convex hull problem, SIAM J. Comput. 20 2 (1991) 259–269.

[10] M.T. Goodrich, Finding the convex hull of a sorted point set in parallel, Inform. Process. Lett. 26 (1987) 173–179.

[11] R.L. Graham, An efficient algorithm for determining the convex hull of a finite planar set, Inform. Process Lett. 1 (1972) 132–133.

[12] R.L. Graham and F.F. Yao, Finding the convex hull of a simple polygon, J. Algorithms 4 (1983) 324–331.

[13] D.G. Kirkpatrick and R. Seidel, The ultimate convex hull algorithm?, SIAM J. Comput. 15 (1986) 287–299.
[14] D.T. Lee and F.P. Preparata, Computational Geometry—A survey, IEEE Trans. Comput. Vol. 33 (December 1984) 872–1101.
[15] N. Megiddo, Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems, SIAM J. Comput. 12 (1983) 759–766.
[16] N. Megiddo, Linear Programming in linear time when the dimension is fixed, J. ACM 31 (1984) 114–127.
[17] F.P. Preparata and M.I. Shamos, Computational Geometry: An Introduction (Springer, Berlin, 1985).
[18] R. Seidel, Linear Programming and Convex Hulls Made Easy, Proc. 6th ACM Symp. on Computational Geometry (1990) 211–215.
[19] A.C. Yao, A lower bound to finding convex hulls, J. ACM 28 (1981) 780–789.