Efficient Computation of the Outer Hull of a Discrete Path*

Srecko Brlek, Hugo Tremblay, Jérôme Tremblay, and Romaine Weber

Laboratoire de Combinatoire et d'Informatique Mathématique,
Université du Québec à Montréal,
CP 8888 Succ. Centre-ville, Montréal (QC) Canada H3C 3P8
{brlek.srecko,jerome.tremblay}@uqam.ca, hugo.tremblay@lacim.ca,
weberomaine@gmail.com

Abstract. We present here a linear time and space algorithm for computing the outer hull of any discrete path encoded by its Freeman chain code. The basic data structure uses an enriched version of the data structure introduced by Brlek, Koskas and Provençal: using quadtrees for representing points in the discrete plane $\mathbb{Z} \times \mathbb{Z}$ with neighborhood links, deciding path intersection is achievable in linear time and space. By combining the well-known wall follower algorithm for traversing mazes, we obtain the desired result with two passes resulting in a global linear time and space algorithm. As a byproduct, the convex hull is obtained as well.

Keywords: Freeman code, lattice paths, radix tree, discrete sets, outer hull, convex hull.

1 Introduction

The ever-growing use of digital screens in industrial, military and civil applications gave rise to a new branch of study of discrete objects: digital geometry, where objects are sets of pixels. In particular, their various geometric properties play an essential role, for allowing the design of efficient algorithms for recognizing patterns and extracting features: these are mandatory steps for an accurate interpretation of acquired images.

Convex objects play a prominent role in several branches of mathematics, namely functional analysis, optimization, probability and mathematical physics (see [1] for a detailed account of convex geometry and applications). In Euclidean geometry, given a finite set of points, the problem of finding the smallest convex set containing all of them led to the introduction of the geometric notion of *convex hull*. On the practical side, the computation of the convex hull proved to be one of the most fundamental algorithm in computational geometry as it has many applications ranging from operational research [2] to design automation [3]. It is also widely used in computer graphics, and particularly in image processing [4]. For example, the Delaunay triangulation of a d-dimensional set of points

* With the support of NSERC (Canada).

E. Barcucci et al. (Eds.): DGCI 2014, LNCS 8668, pp. 122–133, 2014.

in Euclidean space is equivalent to finding the convex hull of a set of $d+1$-dimensional points [5]. It is well known that for the Euclidean case, algorithms for computing the convex hull of a set $S \subset \mathbb{R}^2$ run in $\mathcal{O}(n \log n)$ time where $n = |S|$ (see [6,7]). One can also show that such algorithms are optimal up to a linear constant (see [8,9,10] for the general case).

Nevertheless, by confining the problem to computing the convex hull of simple polygons, linear asymptotic bounds are achieved (see [11,12]). The digital version of this problem is a little more involved. For instance, one can compute the convex hull of a set of pixels S by first computing the Euclidean convex hull of S and then digitalizing the result [13]. This automatically yields $\mathcal{O}(n \log n)$ asymptotical bounds in the worst case. In the discrete case, the situation is surprisingly easier with the help of combinatorics on words, a field which recently led to the development of efficient tools to study digital geometry (see [17,18]). For instance, linear asymptotic bounds are obtained when considering discrete paths encoded by elementary steps. Indeed, Brlek et al. designed a linear time algorithm for computing the discrete convex hull of non self-intersecting closed paths in the square grid [14]. It is based on an optimal linear time and space algorithm for factorizing a word in Lyndon words designed by Duval [15]. The situation is more complicated for intersecting paths.

Here, we describe a linear algorithm for computing the outer hull of any discrete path using the data structure described in [16] where the authors designed a linear time and space algorithm for detecting path intersection. It rests on the encoding of points in the discrete plane $\mathbb{Z} \times \mathbb{Z}$ by quadrees deduced from the radix order representation of binary coordinate points. Then, each path is dynamically encoded by adding a pointer for each step of the discrete path encoded on the four letter alphabet $\{0,1,2,3\}$. Starting from that, the wall follower algorithm used for maze solutions allows to take at each intersection the rightmost available step. The resulting two-passes algorithm is linear in space and time. As a byproduct, the convex hull of any discrete path is computed in linear time.

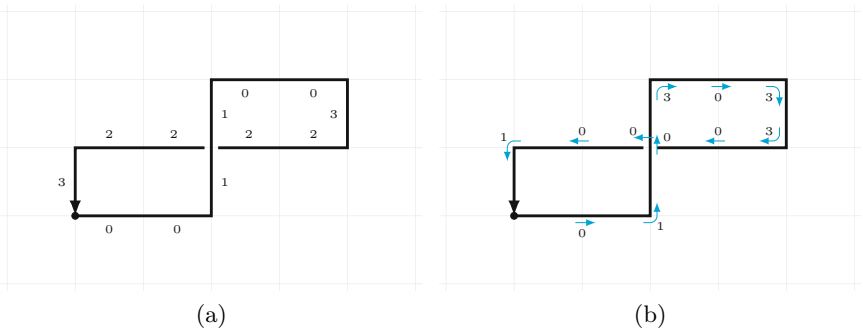
2 Preliminaries

Given a finite alphabet Σ, a *word* w is a function $w : [1, 2, \ldots, n] \longrightarrow \Sigma$ denoted by its sequence of letters $w = w_1 w_2 \cdots w_n$, and $|w| = n$ is its *length*. For $a \in \Sigma$, $|w|_a$ is the number of letters a in w. The set of all words of length k is denoted by Σ^k. Consequently, $\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i$ is the set of all finite words on Σ where $\Sigma^0 = \{\varepsilon\}$, the set consisting of the empty word. Σ^* together with the operation of concatenation form a monoid called *the free monoid on* Σ.

There is a bijection between the set of pixels and $\mathbb{Z}^2$ obtained by mapping $(a, b) \in \mathbb{Z}^2$ to the unitary square whose bottom left vertex coordinate is (a, b). Therefore, we may consider pixels as elements of $\mathbb{Z}^2$. By definition, a *discrete set* S is a set of pixels, i.e. $S \subset \mathbb{Z}^2$. Also, S is called 4-*connected* if each pair of pixels share a common edge and 8-*connected* if each pair of pixels share a common edge or vertex. Since any discrete set is a disjoint collection of 8-connected sets, we consider from now on that discrete sets are 4 or 8-connected.

A convenient way of representing discrete sets without hole is to use a word describing its contour. In 1961, Herbert Freeman proposed an encoding of discrete objects by specifying their contour using the four elementary steps $(\rightarrow, \uparrow, \leftarrow, \downarrow) \simeq (0, 1, 2, 3)$ [19]. This encoding provides a convenient representation of discrete paths in $\mathbb{Z}^2$. By definition, a *discrete path* P is a sequence of points $P = \{p_1, p_2, \ldots, p_n\}$ where p_i and p_{i+1} are neighbors for $1 \leq i < n$. Intuitively, two points u and v are neighbors if and only if $u = v \pm e$ where $e \in \{(1, 0), (0, 1), (-1, 0), (0, -1)\}$.

It is clear from these definitions that any discrete path P is represented by a word $w \in \mathcal{F}^*$ where $\mathcal{F} = \{0, 1, 2, 3\}$ is the Freeman alphabet. It is worth mentioning that in the case of a closed discrete path, w is unique up to a circular permutation of its letters. For example, any circular permutation of the word $w = 001100322223$ represents the discrete path shown in Figure 1(a). One says that a word $w \in \mathcal{F}^*$ is *closed* if and only if $|w|_0 = |w|_2$ and $|w|_1 = |w|_3$. Further, w is called *simple* if it codes a non self-intersecting discrete path. For instance, $w = 001100322223$ is non-simple and closed.



(a) (b)

Fig. 1. (a) A discrete path coded by the word $w = 001100322223$; (b) and its first difference word $\Delta(w) = 01030330001$

It is sometimes useful to consider encoding of paths with turns instead of elementary steps. Such encoding is obtained from the contour word $w = w_1 \cdots w_n$ by setting

$$\Delta(w) = (w_2 - w_1)(w_3 - w_2) \cdots (w_n - w_{n-1})$$

where subtraction is computed modulo 4. $\Delta(w)$ is called the *first differences word of* w. Letters of $\Delta(w) \in \mathcal{F}^*$ are interpreted via the bijection $(0, 1, 2, 3) \simeq$ (forward, left turn, u-turn, right turn). For example, one can verify in Figure 1(b) that $\Delta(w) = 01030330001$ and that it codes the turns of w.

Now, every path w is contained in a smallest rectangle, or bounding box such that we can define the point W as in Figure 2(a). W is easily obtained in linear time by keeping track of the extremum coordinates while reading the word. It is worth mentioning that in the case of a closed simple path u, this coordinate

corresponds to the point W of the standard decomposition of u obtained by considering the following four extremal points of the bounding box: W (lowest on the left side), N (leftmost on the top side), E (highest on the right side) and S (rightmost on the bottom side) (see Figure 2(b)).

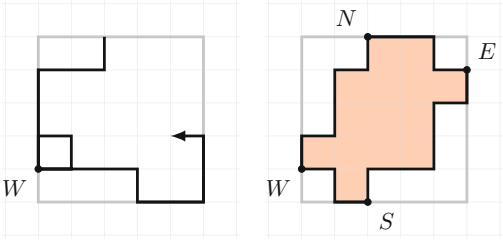


Fig. 2. (a) Smallest rectangle containing a discrete path and the point W; (b) Standard decomposition of a self-avoiding closed path

We close this section by recalling some notions about topological graph theory (see [20] for a thorough exposition of the subject). Let P be a discrete path (i.e. a sequence of integer points) coded by the word w. The image of P as a subset of $\mathbb{R}^2$ is noted G_P while the graph of its image embed in the plane $\mathbb{R}^2$ is noted $\mathcal{G}(P)$. Such embeddings in the plane are completely determined by associating a cyclic order on the edges around each vertex in the following way: Begin by fixing an orientation at each point (e.g. counterclockwise). Then, for each vertex v in G_P, define the cyclic permutation on incident edges of v. This defines a rotation scheme on G_P. One can then show that such a scheme is equivalent to an oriented embedding of G_P on a surface. For example, Figure 3 illustrates the path P coded by $w = 001233$, its graph G_P and its associated counterclockwise embedding $\mathcal{G}(P)$ in $\mathbb{R}^2$.

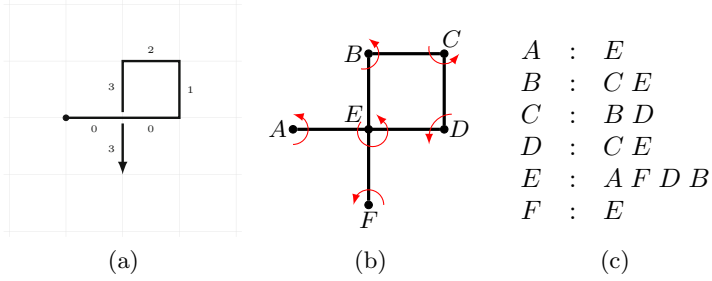


Fig. 3. (a) The graph G_P associated to the path coded by $w = 001233$; (b) The counterclockwise embedding $\mathcal{G}(P)$ in $\mathbb{R}^2$; (c) And its associated rotation scheme

3 Outer and Convex Hull

We recall from topology that given a set S, the boundary ∂S is the set of points in the closure of S, not belonging to the interior of S. Now, let S be a 8-connected discrete set. The *outer hull of* S, denoted Hull(S) is the boundary of the intersection of all discrete sets without hole containing S, i.e. the non self-intersecting path following the exterior contour of S. Definition 1 extends the notion of outer hull to any discrete path.

Definition 1. *Let P be any discrete path. Then, the outer hull of P, denoted by* Hull(P) *is the outer face of the embedded graph $\mathcal{G}(P)$.*

The difference between Definition 1 and the preceding one lies in the use of the embedding of P in the plane instead of a discrete set to describe the outer hull. This choice is not arbitrary as it allows the treatment of discrete line segments (i.e. Euclidean sets of area 0). For example, Figure 4 illustrates the outer hull of the path coded by $w = 021$. Remark that using Definition 1, the boundary of discrete line segments are coded by closed words, e.g. the outer hull of the horizontal line segment coded by 0 is coded by 02 (see Figure 4). This ensures that Definition 1 is a convenient generalization of the outer hull to discrete paths. Indeed, if P codes the boundary of a discrete set S, then P is simple and closed by definition. This gives $P = \text{Hull}(P)$ and since Hull(S) is the boundary $\partial(S)$ of S by definition, we have

$$\text{Hull}(S) = \partial(S) = P = \text{Hull}(P).$$

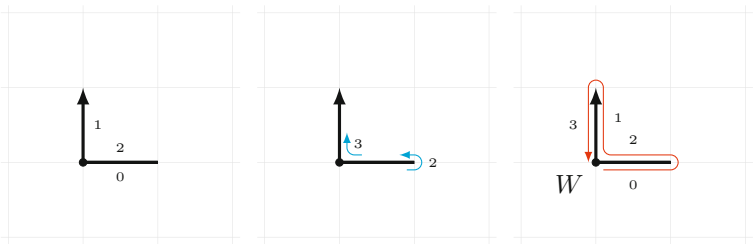


Fig. 4. (a) The path $w = 021$, (b) its first diference word $\Delta(w) = 23$ and (c) its outer hull Hull(w) = 0213

Since there is a bijection between discrete paths in $\mathbb{Z}^2$ and words on $\mathcal{F}$, we identify P with its coding word w and we write Hull(w) instead of Hull(P).

Finally, we recall some basic notions concerning digital convexity, for which a detailed exposure appears in [14,18]. Let S be an 8-connected discrete set. S is *digitally convex* if it is the Gauss digitalization of a convex subset R of $\mathbb{R}^2$, i.e. $S = \text{Conv}(R) \cap \mathbb{Z}^2$. The *convex hull of* S, denoted Conv(S) is the intersection of all convex sets containing S. In the case of a closed simple path w, Conv(w) is

given by the Spitzer factorization of w (see [21,14]). Given $w = w_1 w_2 \cdots w_n \in \{0,1\}^*$, one can compute the NW part of this factorization as follows: Start with the list $(b_1, b_2, \ldots, b_n) = (w_1, w_2, \ldots, w_n)$. If the slope $\rho(b_i) = |b_i|_1/|b_i|_0$ of b_i is strictly smaller than that of b_{i+1} for some i, then

$$(b_1, b_2, \ldots, b_k) = (b_1, \ldots, b_{i-1}, b_i b_{i+1}, b_{i+2}, \ldots, b_k).$$

By repeating this process until it is no longer possible to concatenate any words, one obtains the Spitzer factorization of w. The NE, SE and SW parts of the factorization are obtained by rotations.

4 Algorithm

Let $w \in \mathcal{F}^*$ be a discrete path and G_w its graph representation. Remark that the application $g : w \mapsto G_w$ is not bijective since it is not injective (for example, $u = 0$ and $v = 02$ admits the same graph). Now, recall from Section 2 that the embedding $\mathcal{G}(w)$ of G_w in $\mathbb{R}^2$ gives rise to a rotation scheme (provided we fix an orientation). We use this embedding to compute the outer hull of w (i.e. the outer face of $\mathcal{G}(w)$): Fix an orientation $\mathcal{O}$ of the surface $\mathbb{R}^2$ and let $e_i = (u, v)$ be an arc from vertex u to v in $\mathcal{G}(w)$ such that e_i is an edge of the outer face of the embedding $\mathcal{G}(w)$ and such that e follows the fixed orientation $\mathcal{O}$. Next, compute $e_{i+1} = (v, \sigma_v(u))$ where σ_v is the cyclic permutation associated to v in $\mathcal{G}(w)$. By letting $\mathcal{O}$ be the counterclockwise orientation, one can iterate this process to obtain the outer face of $\mathcal{G}(w)$. For example, using the rotation scheme defined for $w = 001233$ in Figure 3(c) and starting with the arc (A, E), one computes the sequence of arcs

$$(A,E),\ (E,F),\ (F,E),\ (E,D),\ (D,C),\ (C,B),\ (B,E),\ (E,A)$$

which corresponds to the outer hull of w (see Figure 5).

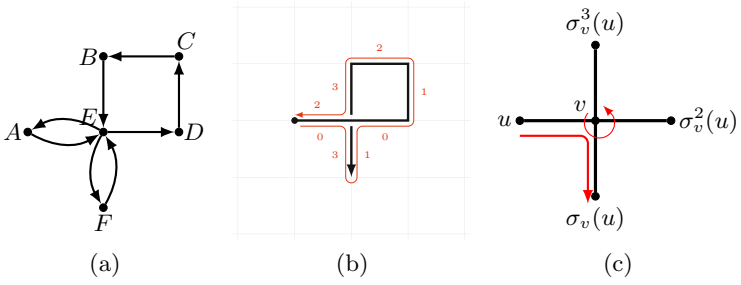


Fig. 5. (a) The sequence of arcs obtained by using the rotation scheme of Figure 3(c); (b) The outer hull of $w = 001233$; (c) The sequence (u, v), $(v, \sigma_v(u))$ corresponds to a right turn in the graph of a path, provided the orientation is counterclockwise

The correctness of this method follows from the so-called "right-hand rule" or "wall follower algorithm" for traversing mazes. Indeed, given an arc (u, v), taking the adjacent arc $(v, \sigma_v(u))$ amounts to "turning right" at vertex v (see Figure 5(c)). The underlying principle of our algorithm is thus to walk along the path, starting at an origin point on the outer hull and turning systematically right at each intersection and returning to the origin point. The preceding discussion guarantees that the resulting walk is then precisely the outer hull of w.

To efficiently implement this procedure, several problems must be addressed. First, as stated before, the walk needs to start on a coordinate of the outer hull, otherwise the resulting path may not describe the correct object. This can be solved by choosing the point W associated with the contour word w as the starting point.

Secondly, whenever a path returns to W (the simplest of which is the path coded by $w = 021$, see Figure 4), before continuing on, one must make sure that the algorithm does not stop until every such sub-path has been explored. An easy solution for managing that situation is to keep a list of all neighbors of W that are in the path P associated with w. This list has at most two elements since no vertex in P is located below or left of W.

Finally, one needs to recognize intersections and decide of the rightmost turn. We solve this problem by using a quadtree structure keeping information on neighborhood relations. This so-called radix quadtree structure was first introduced by Brlek, Koskas and Provençal in [16] for detecting path intersections. Given a discrete path w starting at $(x, y) \in \mathbb{N}^2$ and staying in the first quadrant, the *quadtree structure associated to* w (see [18] and [16] for the generalization to all four quadrants) is described as follows. $G = (N, R, T)$ is a quadtree where:

N is the set of vertices associated to points in the plane;
R is a set of edges representing the fatherhood relation: $r \in R$ is an edge from (x, y) to $(x', y') \iff (x', y')$ is a child of (x, y), that is if $(x', y') = (2x + \alpha, 2y + \beta)$ where $(\alpha, \beta) \in \{0, 1\}^2$;
T is a set of edges representing the neighborhood relation: $t \in T$ is an edge from (x, y) to $(x', y') \iff (x', y')$ is a neighbor of (x, y), that is if $(x', y') = (x, y) + \boldsymbol{e}$ where $\boldsymbol{e} \in \{(1, 0), (0, 1), (-1, 0), (0, -1)\}$ (see Example 2).

One should note that the quadtree structure is described in [16] with unidirectional edges while in this paper all edges are considered bidirectional. Further, by following the procedure described in [16] to build the quadtree, one adds neighborhood links between non-visited nodes during the recursion process. This is easily fixed by adding a boolean label to each neighborhood edge indicating if that specific edge is part of the discrete path or if it has been added by a recursive call. This ensures that the points u and v are neighbors if and only if there is a non-labeled neighborhood edge between these two vertices in the quadtree structure.

It is worth mentioning that this structure is computed in linear time and space. Moreover, it can be generalized to any discrete path as opposed to paths staying in the first quadrant.

Example 2. *Let* $w = 001100322223$ *be the word coding the discrete path in Figure 1 translated to the origin. The quadtree structure associated to* w *is represented in Figure 6. Parenthood and neighborhood relations are respectively represented by black and red edges. Visited nodes are marked by red squares.*

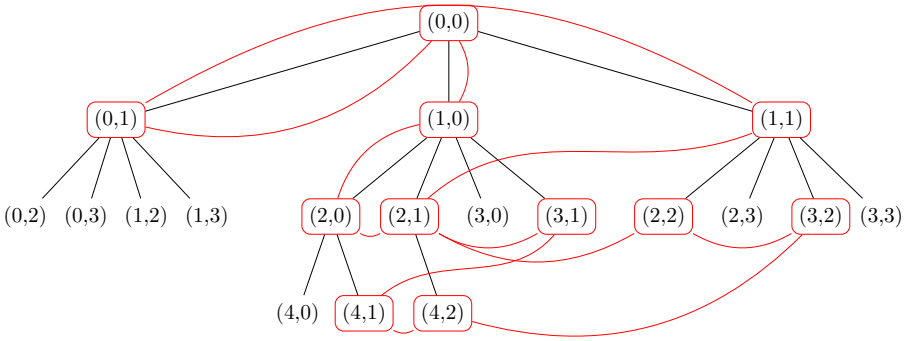


Fig. 6. Quadtree corresponding to the word $w = 001100322223$. Neighborhood edges added by recursive calls are omitted.

This gives rise to the following Algorithm 1 to compute the outer hull of a discrete path w, which proceeds as follows.

Algorithm 1. Outer hull

Require: A word $w \in \mathcal{F}^*$ coding a discrete path
Ensure: A simple word $w' \in \mathcal{F}^*$ describing Hull(w)

Construct the quadtree G associated to w rooted in W
Let $\mathtt{W}$ be the leftmost lowest coordinate on the bounding box of w
Let $\mathtt{N}$ be the set of all visited neighbors of W
$\mathtt{c} \leftarrow \mathtt{W} + (1, 0)$ if it is in $\mathtt{N}$ or $\mathtt{W} + (0, 1)$ otherwise
$w' = \text{Step}(\mathtt{c} - \mathtt{W})$
while $\mathtt{c} \neq \mathtt{W}$ or $\mathtt{N} \neq \varnothing$ **do**
    $\mathtt{turn} = 2 \mod 4$
    for each neighbor $\mathtt{v}$ of $\mathtt{c}$ **do**
        if $[\text{Step}(\mathtt{v} - \mathtt{c}) - \text{Lst}(w')] + 1 \mod 4 \leq [\mathtt{turn}] + 1 \mod 4$ **then**
            $\mathtt{turn} \leftarrow \text{Step}(\mathtt{v} - \mathtt{c}) - \text{Lst}(w')$
            $\mathtt{next} \leftarrow \mathtt{v}$
        end if
    end for
    $w' = w' \cdot \text{Step}(\mathtt{next} - \mathtt{c})$
    remove $\mathtt{c}$ from $\mathtt{N}$
    $\mathtt{c} \leftarrow \mathtt{next}$
end while
return w'

It is assumed that the point W, that is the leftmost lower point of the bounding box is known. First, the quadtree G associated to w is built starting from W. Then, the graph G is traversed from its root W, following the path represented by w. At every intersection c, we need to:

(a) extract the letter α associated to the vector $\overrightarrow{cv}$ for each neighbor v of c;
(b) determine the turn associated to each v, that is $\Delta(w_c \cdot \alpha)$;
(c) choose the rightmost one, that is the closest to 3.

This procedure ends when returning to the point W.

Theorem 3 (Correctness of Algorithm 1). *For any word $w \in \mathcal{F}^*$, Algorithm 1 returns* Hull(w).

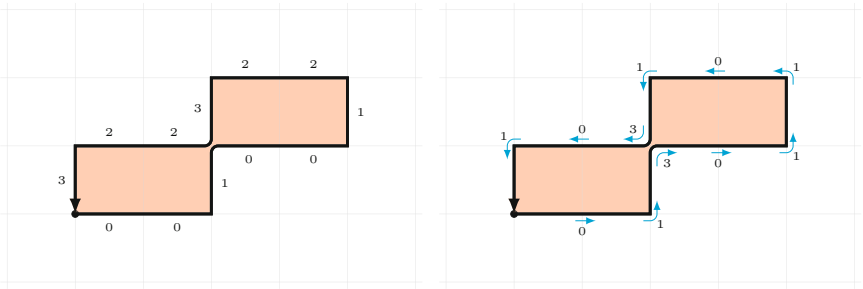
Proof. Let Hull(w) be of length $k \in \mathbb{N}^+$. We use the following loop invariant:

> At the start of the i^{th} iteration of the while loop in Line 6, w' is a prefix of length i of the contour word associated to Hull(w).

The invariant holds the first time Line 6 is executed, since at that time, w' is the first step of the outer hull of w computed at Line 5. Now, assume the invariant holds before the i^{th} iteration of the loop. Then, Lines 8 to 13 find the rightmost turn at the current coordinate `c`. Then in Line 14, w' is concatenated with the step of this turn. By the right-hand rule for solving simply connected maze, considering rightmost turns yields coordinates on the outer hull of w. Consequently, at the end of the iteration, w' is a prefix of the contour word associated to Hull(w) of length $i+1$. Finally, at the end of the loop, w' is a prefix of the contour word associated to Hull(w) of length k, that is $w' =$ Hull(w). Note that since any neighbor of W is on Hull(w), Line 15 clearly removes every element from `N` yielding, at termination, an empty set. □

We end this section by showing that Algorithm 1 is linear in time and space. First, the quadtree structure is constructed in linear time (see [16]). Also, as stated before, the point W is easily computed in linear time. Consequently, computations in Line 1 are performed in linear time. Next, Line 2, 4 and 5 each take constant time. Moreover, the set `N` is constructed in linear time by accessing neighborhood informations of the root in the quadtree structure, so Line 3 takes linear time. Now, since any coordinate has at most four neighbors, the **for** loop in Line 8 is executed at most four time per iteration of the **while** loop. Line 15 takes constant time. This is due to the fact that `N` contains at most two elements. Since instructions in Line 7, 9, 10, 11, 14 and 16 all are computed in constant time, at most $k(4c_1 + c_2)$ computations occur during the execution of the **while** loop where $k \in \mathbb{N}^+$ is the length of Hull(w) and $c_1, c_2 \in \mathbb{R}$ some constants, thus making Algorithm 1 linear in time. Finally, the quadtree structure needs space linear in the length of the path, so that our algorithm is also linear in space.

Example 4. *Consider the word* $w = 001100322223$ *of Example 2. Then, Algorithm 1 yields* $w' = 001001223223$ *(see Figure 7). One can easily verify that* w' *is a simple path describing the outer hull of* w*, so* $\text{Hull}(w) = w'$.



(a) $w' = 001001223223$ (b) $\Delta(w') = 01301101301$

Fig. 7. Outer hull of $w = 001100322223$

Our algorithm was implemented using the C++ programming language and tested with numerous examples (see Figure 8). The source code is available at `http://bitbucket.org/htremblay/outer_hull`.

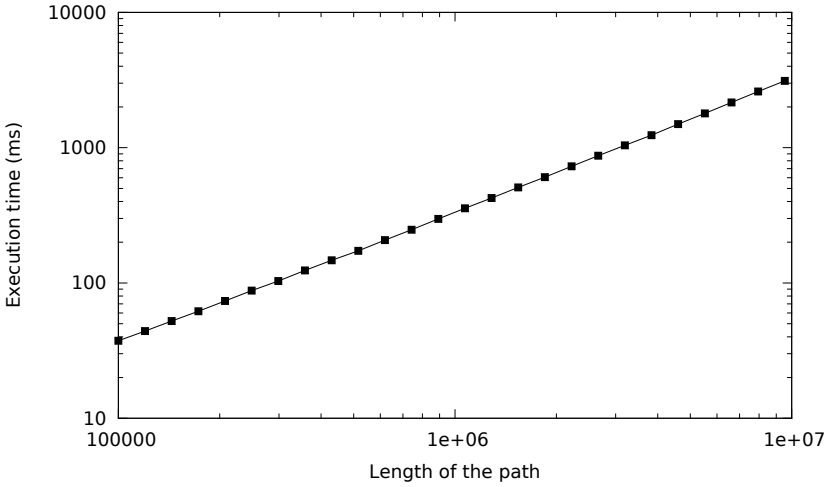


Fig. 8. Running time of Algorithm 1 for random discrete paths of length 10^5 to 10^7, with each point representing the mean running time of 100 random discrete paths of same length

Finally, we show how Algorithm 1 can be used to compute in linear time and space the convex hull of any discrete path. It relies on the following rather obvious result:

Proposition 5. *Let $w \in \mathcal{F}^*$ be a boundary word coding a discrete path. Then,*

$$\mathrm{Conv}(w) = \mathrm{Conv}(\mathrm{Hull}(w)).$$

Proof. If w is simple, then $\mathrm{Hull}(w) = w$ so the claim holds. Now, suppose w is non-simple. Then by definition, $\mathrm{Hull}(w)$ is the boundary of w. Since, $\mathrm{Conv}(w)$ is the intersection of all convex sets containing w, it must also contain $\mathrm{Hull}(w)$ and thus $\mathrm{Conv}(w) = \mathrm{Conv}(\mathrm{Hull}(w))$. □

Recall that $\mathrm{Hull}(w)$ is non self-intersecting for any path w. Proposition 5 then yields a very simple procedure for computing the convex hull of a discrete path using Brlek et al. simple path convex hull algorithm (see [14]):

1. Start by computing $\mathrm{Hull}(w) = w'$;
2. Compute $\mathrm{Conv}(w')$.

It is clear that the preceding procedure computes the convex hull of a discrete path in linear time and space. Indeed, we showed in Section 4 that the first step is computed in linear time and space. Furthermore, it is shown in [14] that the second step is computed in a similar fashion.

5 Concluding Remarks

We presented an algorithm for computing the outer hull of a discrete path. This led to a procedure for computing the convex hull of any discrete set. Our algorithm is a significant improvement over the convex hull algorithm presented in [14] in the sense that computations can be made on any discrete path as opposed to non self-intersecting ones. Moreover, we proved that such computations can be made in linear time and space.

Instead of computing the outer hull of a discrete path P as described in this paper, one could want to compute the largest simply connected isothetic polygon such that all integers points on its boundary are visited by P. Although some modifications to our algorithm are necessary in order to perform such computations, the time complexity would not change.

In addition, this research begs to be generalized to three dimensional discrete spaces, that is geometry in Euclidean space $\mathbb{R}^3$ studying sets of unit cubes. Also, applications of our algorithm is not limited to convex hull problems. We plan on using it to study various path intersections problems such as primality, union, intersection and difference of discrete sets.

Acknowledgement. Martin Lavoie helped in implementing Algorithm 1. Thanks Martin. We are also grateful to the reviewers for the accurate comments which substantially improved the theoretical background of our work.

References

1. Gruber, P.M.: Convex and discrete geometry. Springer (2007)
2. Sherali, H., Adams, W.: A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. SIAM Journal on Discrete Mathematics 3(3), 411–430 (1990)
3. Kim, Y.S.: Recognition of form features using convex decomposition. Computer-Aided Design 24(9), 461–476 (1992)
4. Kim, M.A., Lee, E.J., Cho, H.G., Park, K.J.: A visualization technique for DNA walk plot using k-convex hull. In: Proceedings of the Fifth International Conference in Central Europe in Computer Graphics and Visualization, Plzeň, Czech Republic, Západočeská univerzita, pp. 212–221 (1997)
5. Okabe, A., Boots, B., Sugihara, K.: Spacial tesselations: Concepts and applications of Voronoi diagrams. Wiley (1992)
6. Graham, R.A.: An efficient algorithm for determining the convex hull of a finite planar set. Information Processing Letters 1(4), 132–133 (1972)
7. Chan, T.M.: Optimal output-sensitive convex hull algorithms in two and three dimensions. Discrete & Computational Geometry 16, 361–368 (1996)
8. Yao, A.C.C.: A lower bound to finding the convex hulls. PhD thesis, Stanford University (April 1979)
9. Chazelle, B.: An optimal convex hull algorithm in any fixed dimension. Discrete & Computational Geometry 10, 377–409 (1993)
10. Goodman, J.E., O'Rourke, J.: Handbook of discrete and computational geometry, 2nd edn. CRC Press (2004)
11. McCallum, D., Avis, D.: A linear algorithm for finding the convex hull of a simple polygon. Information Processing Letters 9(5), 201–206 (1979)
12. Melkman, A.: On-line construction of the convex hull of a simple polyline. Information Processing Letters 25, 11–12 (1987)
13. Chaudhuri, B.B., Rosenfeld, A.: On the computation of the digital convex hull and circular hull of a digital region. Pattern Recognition 31(12), 2007–2016 (1998)
14. Brlek, S., Lachaud, J.O., Provençal, X., Reutenauer, C.: Lyndon + Christoffel = digitally convex. Pattern Recognition 42, 2239–2246 (2009)
15. Duval, J.P.: Factorizing words over an ordered alphabet. J. Algorithms 4(4), 363–381 (1983)
16. Brlek, S., Koskas, M., Provençal, X.: A linear time and space algorithm for detecting path intersection. Theoretical Computer Science 412, 4841–4850 (2011)
17. Blondin Massé, A.: À l'intersection de la combinatoire des mots et de la géométrie discrète: Palindromes, symétries et pavages. PhD thesis, Université du Québec à Montréal (February 2012)
18. Provençal, X.: Combinatoire des mots, géométrie discrète et pavages. PhD thesis, Université du Québec à Montréal (September 2008)
19. Freeman, H.: On the encoding of arbitrary geometric configurations. IRE Transactions on Electronic Computers EC-10(2), 260–268 (1961)
20. Gross, J.L., Tucker, T.W.: Topological graph theory. Wiley (1987)
21. Spitzer, F.: A combinatorial lemma and its application to probability theory. Transactions of the American Mathematical Society 82, 323–339 (1956)