Las Vegas Gift-Wrapping is Twice as Fast

(Extended Abstract for 2nd CCCG)

Rex A. Dwyer[1]
North Carolina State University

Abstract

A randomized version of the gift-wrapping algorithm for convex hulls performs only about half as many floating-point operations as the standard version for large problems. Furthermore, the floating-point working set of its inner loop is half as large, which magnifies the potential speed-up for certain dimensions on machines with a limited number of floating-point registers. Ongoing work is aimed at analytically proving the conjecture that the randomized algorithm is faster with very high probability. This has been verified numerically for moderately large point sets.

The convex hull of a set of n points in $\mathbf{R}^d$ is the intersection of all closed halfspaces that contain all n points. The gift-wrapping algorithm [1, 2] for constructing a description of the structure of the convex hull of a point set is based on the observation that every subfacet ($(d-2)$-dimensional face) of the convex hull is contained by exactly two facets ($(d-1)$-dimensional faces). The algorithm maintains a list of subfacets for which exactly one containing facet is known. At each step of the algorithm, a subfacet is removed from this list, and the unknown point (or points) completing its unknown facet are determined. The new facet is output. Next, all of its subfacets are searched for in the list. Those found are deleted, since both of their facets are now known. Those not found are inserted; for these the most recently found facet is the only one known. The first facet is found by invoking the procedure recursively in $(d-1)$ dimensions.

The gift-wrapping algorithm is not an optimal algorithm, but it is reasonably fast, conceptually simple, and not difficult to implement. Since it appears to be a good choice in many applications, constant-factor improvements to its running time are desirable so long as they do not make the algorithm more complicated. In this paper we present such an improvement based on randomization techniques. We show that the improved algorithm is twice as fast on average, and faster with high probability. While the analysis of the "faster-with-high-probability" claim is apparently quite difficult, the mean is quite easy, and, at this level, the conceptual simplicity of the algorithm is maintained.

In the sequel, we assume that the n points are in general position. This implies that the convex hull is *simplicial*, that is, that each k-face is a k-simplex defined by exactly $k+1$ points. Now consider a d-tuple of non-coplanar points $F = (F_1, F_2, \ldots, F_d)$ and a further point P. The points of F define a hyperplane and two disjoint open halfspaces; we say that one is *below* F and the other is *above* F. The order of the points in the d-tuple determines which side is below and which above; an even permutation of the points preserves this; an odd permutation reverses it. Also, if P is above F, then F_d is below $(F_1, F_2, \ldots, F_{d-1}, P)$. If a d-tuple of points defines a facet of the convex hull, clearly every other point lies below it,

[1]Supported by National Science Foundation Grant CCR–8908782. Address: Dept. of Computer Science, N.C. State University, Raleigh, NC 27695–8206. E-mail: dwyer@cscadm.ncsu.edu.

Find an initial facet $(F_1, F_2, \ldots, F_d)$;
$InsertQ(F_d, F_2, F_3, \ldots, F_{d-1}; F_1)$; $InsertQ(F_1, F_d, F_3, \ldots, F_{d-1}; F_2)$;
$\ldots InsertQ(F_d, F_2, F_3, \ldots, F_d; F_{d-1})$;

While $\neg EmptyQ()$ do
begin
$(F_1, F_2, \ldots, F_{d-1}; P_{init}) := DeleteQ()$;
$P_{cand} := P_{init}$;
for $i := 1$ to n do
if $Above((F_1, F_2, \ldots, F_{d-1}, P_{cand}), P_i)$ then $P_{cand} := P_i$;
$P_{max} := P_{cand}$;
$InsertQ(P_{max}, F_2, F_3, \ldots, F_{d-1}; F_1)$; $InsertQ(F_1, P_{max}, F_3, \ldots, F_{d-1}; F_2)$;
$\ldots InsertQ(F_d, F_2, F_3, \ldots, P_{max}; F_{d-1})$;
end;

Figure 1: Naïve Gift-Wrapping

and none lies above it. An acceptable solution to the convex-hull problem is an enumeration of the d-tuples that define facets.

If we define the procedure $Above(F, P)$ to return "true" when P is above F, we can express a simple version of the gift-wrapping algorithm by the pseudo-code of Figure 1. We assume that the procedure $InsertQ$ inserts a subfacet record into the list only after searching to see if it is already there, deleting both records if it is found. A subfacet's record in the list is actually an odd permutation of its known facet; thus every other point lies above this d-tuple. The inner loop attempts to replace the last point in the d-tuple, P_{cand} by another so that every point lies below the d-tuple. It does this by examining every point P_i and replacing P_{cand} by P_i if P_i is above the current d-tuple. Every update of P_{cand} as the search progresses increases the angle between the hyperplanes defined by $(F_1, F_2, \ldots, F_{d-1}, P_{init})$ and $(F_1, F_2, \ldots, F_{d-1}, P_{cand})$. Finally, when the inner loop terminates, P_{max} is the point that maximizes this angle.

As is well known, the test $Above((F_1, F_2, \ldots, F_{d-1}, P_{cand}), P_i)$ may be implemented by evaluating

$$\begin{vmatrix} 1 & P_i^{(1)} & P_i^{(2)} & \cdots & P_i^{(d)} \\ 1 & P_{cand}^{(1)} & P_{cand}^{(2)} & \cdots & P_{cand}^{(d)} \\ 1 & F_{d-1}^{(1)} & F_{d-1}^{(2)} & \cdots & F_{d-1}^{(d)} \\ \vdots & & & & \\ 1 & F_2^{(1)} & F_2^{(2)} & \cdots & F_2^{(d)} \\ 1 & F_1^{(1)} & F_1^{(2)} & \cdots & F_1^{(d)} \end{vmatrix} > 0. \qquad (1)$$

Evaluating this determinant requires $\Theta(d^3)$ arithmetic operations by a straightforward method such as Gaussian elimination. Thus, if a given input set has n points and its convex hull has f facets, the *while*-loop requires $\Theta(d^3 fn)$ arithmetic operations. In fact, this term dominates the running time.

Chand and Kapur [1] present a more efficient method to find P_{max}. Their method directly computes the tangent of the angle formed by each point P_i, then simply chooses the point with the largest angle. First, two vectors N and E are computed from $(F_1, F_2, \ldots, F_{d-1}, P_{init})$

in $\Theta(d^3)$ time. Then, the n tangents are computed from the formula

$$\tan\theta_i = (\langle P_i, N\rangle - \langle P_{init}, N\rangle)/(\langle P_i, E\rangle - \langle P_{init}, E\rangle).$$

In the inner loop, 1 division, $2d$ multiplications, 2 subtractions, and $2d-2$ additions are performed, for a total of $4d+1$ operations. (The inner products $\langle P_{init}, N\rangle$ and $\langle P_{init}, E\rangle$ can be computed just once for all i.)

The total number of operations for the *while*-loop is $(4d+1)fn + \Theta(d^3 f)$, a significant improvement in the dominant term. It is advantageous to keep $(2d+2)$ quantities in fast registers for the inner loop: The coordinates of N and E, and the two inner products $\langle P_{init}, N\rangle$ and $\langle P_{init}, E\rangle$.

Our work, however, takes a different tack to speeding up this algorithm, a method using random numbers to give a running time that, for *any fixed* input, does about half as many arithmetic operations as the Chand-Kapur algorithm *on the average*. The average here is taken over the behavior of the algorithm, i.e., over possible sequences of random numbers, and not over any assumed input distribution.

The determinant in (1) can be expanded by minors along the first row to give

$$\begin{vmatrix} P_{cand}^{(1)} & P_{cand}^{(2)} & \cdots & P_{cand}^{(d)} \\ F_{d-1}^{(1)} & F_{d-1}^{(2)} & \cdots & F_{d-1}^{(d)} \\ \vdots & & & \\ F_2^{(1)} & F_2^{(2)} & \cdots & F_2^{(d)} \\ F_1^{(1)} & F_1^{(2)} & \cdots & F_1^{(d)} \end{vmatrix} - P_i^{(1)} \begin{vmatrix} 1 & P_{cand}^{(2)} & \cdots & P_{cand}^{(d)} \\ 1 & F_{d-1}^{(2)} & \cdots & F_{d-1}^{(d)} \\ \vdots & & & \\ 1 & F_2^{(2)} & \cdots & F_2^{(d)} \\ 1 & F_1^{(2)} & \cdots & F_1^{(d)} \end{vmatrix} + \cdots + (-1)^d P_i^{(d)} \begin{vmatrix} 1 & P_{cand}^{(1)} & P_{cand}^{(2)} & \cdots & P_{cand}^{(d-1)} \\ 1 & F_{d-1}^{(1)} & F_{d-1}^{(2)} & \cdots & F_{d-1}^{(d-1)} \\ \vdots & & & & \\ 1 & F_2^{(1)} & F_2^{(2)} & \cdots & F_2^{(d-1)} \\ 1 & F_1^{(1)} & F_1^{(2)} & \cdots & F_1^{(d-1)} \end{vmatrix} > 0.$$

Given the values of the $d+1$ minor determinants, the value of $Above((F_1, F_2, \ldots, F_{d-1}, P_{cand}), P_i)$ can be determined in an additional d multiplications and d additions. These determinants can obviously be computed in $O(d^4)$ time — in fact, $O(d^3)$ is possible. Unfortunately, they cannot be computed once per facet, outside the *for*-loop, since they change whenever the assignment $P_{cand} := P_i$ is performed. However, if u_i denotes the number of executions of this statement to find the ith facet, the *while*-loop requires $2dnf + O(d^3 \sum_{i=1}^{f} u_i)$ operations, with $f \le \sum u \le nf$. This is still $\Theta(d^3 nf)$ in the worst case, a factor of $\Theta(d^2)$ worse than the Chand-Kapur algorithm. But in the best case, it is $\Theta(2dnf + d^3 f)$, better by a factor of about 2. Also, only half as many registers — for the $(d+1)$ minor determinants — could be used to advantage.

Since the *for*-loop essentially searches for the maximum of a sequence of angles, we are reminded that the expected number of intermediate maxima in such a search is about $\ln n$ when every permutation of the ranks of the inputs is equally likely. No such condition holds *a priori* for our input, but we can make it hold by randomly shuffling the points in the array in $O(n)$ time at the beginning of the algorithm. This gives a *while*-loop requiring $2dnf + \Theta(nf + d^3 f \log n)$ operations *on average*. This is a significant improvement.

However, we can make only rather weak statements about the probability that the new algorithm is faster than the Chand-Kapur version. Since the number of updates required for the various maximum searches cannot be assumed to be independent, we are restricted to applying inequalities like Markov's, which implies that the new version is slower with probability less than $2d/(4d+1) \approx 1/2$.

A theoretical — but not practical — solution would be to shuffle the array before *each* maximum search. This would guarantee independence and admit a good bound on the

probability of slow-down that decreases exponentially in the product nf. But depending on the random number generator used, it would also increase the operation count $2dnf$ to perhaps $(2d+3)nf$, and add many $(\Omega(nf))$ new memory references as the data is moved during shuffling.

Instead, we explore a less costly method of randomization. We shuffle completely only before the first search. On subsequent searches, we choose a random integer j $(1 \leq j \leq n)$, then, in two successive inner *for*-loops, consider first the jth through nth, and then the first through $(j-1)$st points as candidates. This does not guarantee independence of the number of updates for the maximum searches, but it appears to reduce the maximum possible covariance of updates of successive searches so that non-trivial upper bounds on the variance and higher moments of the total number of updates can be derived. These bounds should be sufficient to prove that the probability of the randomized algorithm being slower that the Chand-Kapur algorithm decreases exponentially in nf, a very strong result.

To analyze the covariance, we are considering the following combinatorial problem: For any permutation Π on $\{1,\ldots,n\}$, let $F(\Pi)$ be the number of running maxima in a left-to-right scan of Π. On the average, we know $EF(\Pi) \sim H_n \sim \ln n$. Call permutations Π_1 and Π_2 *rotationally equivalent* if for some $\nu < n$, we have $\Pi_1(j) = \Pi_2((j+\nu) \bmod n)$ for all j. Let p_{nh} be the probability that a randomly chosen permutation lies in an equivalence class S for which $\max_{\Pi \in S} F(\Pi) \leq h$. The p_{nh} can be shown to satisfy

$$\begin{aligned} p_{1h} &= 1; \\ p_{n1} &= 0 \qquad (n > 1); \\ p_{nh} &= \frac{1}{n-1} \sum_{i=1}^{n-1} p_{i,h-1} p_{n-i,h}. \end{aligned}$$

If we define the generating functions $P_h(z) = \sum_n p_{nh} z^{n-1}$, then

$$\begin{aligned} P_1(z) &= 1; \\ P_{h+1}(z) &= \exp\left(\int P_h(z)\, dz\right) \end{aligned}$$

At present we are investigating this functional equation. Direct numerical calculations based on the recurrence indicate that $E(\max_{\Pi \in S} F(\Pi)) \sim 2H_n$ for a randomly chosen equivalence class S. An analytical proof should follow soon.

References

[1] D. R. Chand and S. S. Kapur. An algorithm for convex polytopes. *Journal of the ACM*, 17(1):78–86, January 1970.

[2] G. Swart. Finding the convex hull facet by facet. *J. Algorithms*, 6:17–48, 1985.