A variant of backwards analysis applicable to order-dependent sets*

Evanthia Papadopoulou[1] and Martin Suderland[2]

1 **Faculty of Informatics, Università della Svizzera italiana (USI), Lugano, Switzerland**
`evanthia.papadopoulou@usi.ch`

2 **Courant Institute, New York University (NYU), New York, NY, USA**
`martin.suderland@cs.nyu.edu`

Abstract

Backwards analysis is a simple, yet powerful technique used in analyzing the expected performance of randomized algorithms: a randomized incremental algorithm is seen as if it were running backwards in time, from output to input [Seidel 1993]. To be applicable, a key requirement is that the algorithm's output is independent of the randomization order. This needs to hold even for intermediate structures, assuming the same set of elements have been processed. In this note we illustrate a variant, which can be applied to algorithms with order-dependent output. As an example, we use the *randomized incremental triangulation* of a point set in $\mathbb{R}^d$, a generalization of Quicksort in higher dimensions, which has been cited by Seidel as a negative example, where backwards analysis could not be applied. We prove that the expected running time of this algorithm is $O(n \log n)$. This variant of backwards analysis was introduced by [Junginger and Papadopoulou, DCG 2023].

1 Introduction

Backwards analysis was popularized in Computational Geometry by Seidel [7]. It is a simple, yet powerful technique to analyze the expected performance of a randomized algorithm. Backwards analysis is based on the observation that the cost of the last step of an algorithm can be often expressed as a function of the complexity of the final output; and thus, the algorithm can be analyzed as if it were running backwards in time, from output to input [7].

In computational geometry, the first algorithm analyzed by backwards analysis was the construction of the Delaunay triangulation of a set of points in convex position in the plane [2]. Since then, backwards analysis has been applied to a plethora of problems, and has become a standard trick in analyzing the expected performance of randomized algorithms, see e.g., [1] and references therein. This includes a simplified analysis of the influential randomized incremental construction paradigm, introduced by Clarkson and Shor [3], and a particularly simple approach to analyze Quicksort, both presented by Seidel in [7]. A key requirement to apply backwards analysis, however, is that the algorithm's output is independent of the randomization order. To illustrate this fact, Seidel pointed out a negative example, where backwards analysis could not be applied. This is the *randomized incremental triangulation* of a point set in $\mathbb{R}^d$, which generalizes Quicksort in higher dimensions. Seidel concluded the section with an open problem: "It remains to be seen whether for fixed $d > 1$ the expected running time of this triangulation algorithm is indeed $O(n \log n)$".

* This research was supported by the Swiss National Science Foundation, Projects 200021E_201356 (Evanthia Papadopoulou) and P500PT_206736/1 (Martin Suderland).

40th European Workshop on Computational Geometry, Ioannina, Greece, March 13–15, 2024.
This is an extended abstract of a presentation given at EuroCG'24. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

Contribution. In this paper we advertise a new variant of backwards analysis, which indeed can prove the aforementioned $O(n \log n)$ time complexity for the randomized incremental triangulation algorithm. Problems where the final output or intermediate structures depend on the randomization order can benefit from this approach. It was first described in [5] for analyzing a randomized incremental algorithm to perform deletion in abstract Voronoi diagrams in linear time The intermediate *Voronoi-like diagrams* computed by this algorithm were order-dependent and thus ordinary backwards analysis could not be correctly applied[1]. In this paper we present another example where this new variant of backwards analysis can analyse the running time of a randomized incremental algorithm while ordinary backwards analysis cannot be applied.

2 Backwards analysis

Preliminaries Given $n \in \mathbb{N}$, we write $[n]$ for $\{1, 2, ..., n\}$. Let S_n be the set of all permutations of length n. We use the one-line notation of permutations.

For a permutation $\sigma = (\sigma_1, \sigma_2..., \sigma_n)$ and index $1 \leqslant i \leqslant n$ let $\sigma(i) = \sigma_i$. Further, we define the *shift permutation* $\sigma^{(i)} = (\sigma_1, ..., \sigma_{i-1}, \sigma_{i+1}, ..., \sigma_n, \sigma_i)$, which moves the i-th element to the end.

The standard backwards analysis is based on the following idea: "Analyze an algorithm as if it were running backwards in time, from output to input. This is based on the observation that often the cost of the last step of an algorithm can be expressed as a function of the complexity of the final product output of the algorithm" [7].

The expected running time of a randomized algorithm is computed step by step. We assume that the algorithm is randomized over a set of n elements. Denote by T_i the running time expended for the i-th step and by O_i the computed structure at step $1 \leqslant i \leqslant n$. Then $T_i(\sigma)$, the time required for the i-th step for permutation σ, is getting bounded by a function of the complexity of the output object $O_i(\sigma)$).

Considering that each permutation is equally likely, we write for the expected time $E(T_i)$:

$$E(T_i) = \frac{\sum_{\sigma \in S_i} T_i(\sigma)}{i!} = \frac{\sum_{j=1}^{i} \sum_{\substack{\sigma \in S_i \\ \sigma(i)=j}} T_i(\sigma)}{i!}.$$

Because the standard backwards analysis is used for order-independent structures, the time needed for the i-th step does not depend on the entire permutation σ but only on the last processed element, i.e. $\sigma(i) = j$.

The variant of backwards analysis exploits the key idea of looking at groups of similar permutations, where the order of elements is almost the same. A group has one representative permutation σ and consists of $G_\sigma = \{\sigma^{(1)}, \sigma^{(2)}, ..., \sigma^{(n)}\}$, where $\sigma^{(n)} = \sigma$.

Essentially each group has one representative from which all other permutations can be derived, by moving one element to the end of the one-line representation, see Fig. 1. Let $R_n \subset S_n$ be a set of $(n-1)!$ many representative permutations such that the groups of representatives is a partition of all permutations of length n, i.e. $\bigcup_{\sigma \in R_n} G_\sigma = S_n$.

[1] The preliminary version of this paper [4] had originally assumed otherwise, after expressing the time complexity of each step as a function of the output structure. The new variant in [5] successfully completed the analysis.

$$G_\sigma = \left\{ \begin{array}{l} \sigma = (\sigma_1, \sigma_2, \sigma_3, ..., \sigma_{n-1}, \sigma_n) = \sigma^{(n)} \\ \sigma^{(1)} = (\sigma_2, \sigma_3, ..., \sigma_{n-1}, \sigma_n, \sigma_1) \\ \sigma^{(2)} = (\sigma_1, \sigma_3, ..., \sigma_{n-1}, \sigma_n, \sigma_2) \\ ... \\ \sigma^{(i)} = (\sigma_1, ..., \sigma_{i-1}, \sigma_{i+1}..., \sigma_n, \sigma_i) \\ ... \\ \sigma^{(n-1)} = (\sigma_1, \sigma_2, ..., \sigma_{n-2}, \sigma_n, \sigma_{n-1}) \end{array} \right\}$$

Figure 1 The group G_σ for representative $\sigma = (\sigma_1, \sigma_2, \sigma_3, ..., \sigma_{n-1}, \sigma_n)$.

For brevity we write $T_i(G_\sigma) = \sum_{\rho \in G_\sigma} T_i(\rho)$. Thus, for the expected time needed for step i we can derive:

$$E(T_i) = \frac{\sum_{\sigma \in S_i} T_i(\sigma)}{i!} = \frac{\sum_{\sigma \in R_i} T_i(G_\sigma)}{i!}.$$

The existence of such a set R_n, for all $n \in \mathbb{N}$, was shown by Levenshtein [6] for problems that were not related to backwards analysis or this variant, and this was originally pointed out to us by Stefan Felsner. For $n = 4$, the set

$$\{(1,2,3,4), (2,1,4,3), (3,1,4,2), (3,2,4,1), (4,1,3,2), (4,2,3,1)\}$$

can be chosen as R_4. Only the existence of a set R_n is important for the variant to work without having to know a particular representation.

Why choose this grouping? The grouping is chosen with the following ideas in mind.

- Each element appears exactly once as last element: this is important for computing the overall expected running time, where each element should appear as last the same number of times.
- The grouping minimizes the number of inversions between the base permutation σ and the permutations within its group G_σ.

Among all groupings satisfying the first property, the one that we chose with G_σ induces the least number of inversions. Minimizing inversions is desirable because it can drastically simplify the derivation compared to other alternative groupings that may satisfy the first item, such as the one generated by swapping elements $\sigma^{(i)}$ and $\sigma^{(n)}$.

To evaluate $T_i(G_\sigma)$ we try to bound each $T_i(\sigma^{(j)})$ by a portion of the output structure $O_i(\sigma)$ for the base permutation σ of the group. We thus evaluate $T_i(G_\sigma)$ as a function involving $O_i(\sigma)$. A minimum number of inversions between $\sigma^{(j)}$ and σ is hense essential for simplifying this task. For our triangulation example this is done in Lemma 3.2.

3 Example: Triangulation algorithm

Given a set of n points $P = \{p_1, p_2, ..., p_n\} \subset \mathbb{R}^d$ for some $d \in \mathbb{N}$ within a d-simplex Δ. We are recalling a higher-dimensional triangulation algorithm of P within Δ, see [7]. Initially, the triangulation T consists of just one simplex Δ. The points are getting inserted one-by-one while we keep the triangulation updated. If point p_i is inserted we are looking for the simplex τ of T, which contains p_i. We replace this simplex τ by the $d + 1$ many simplices, which have p_i as corner each and which partition τ. An overview of the algorithm is given in Algorithm 1 together with an example in Fig. 2. The randomized version is achieved by

Algorithm 1: Triangulating a point set

Input : Set of n points $P = \{p_1, p_2, ..., p_n\}$ within simplex $\Delta \subset \mathbb{R}^d$
Output : Triangulation T of P and Δ

```
T ← Δ;
for i = 1 → n do
    Remove simplex τ, which contains p_i, from T;
    Partition τ into d + 1 simplices, each having p_i as corner;
    Add the new d + 1 simplices to T;
return T;
```

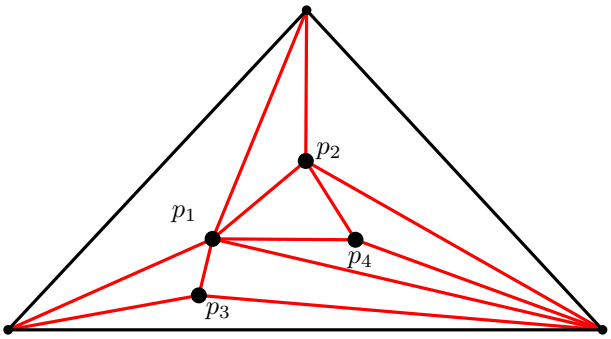

Figure 2 Triangulation of Algorithm 1 for the insertion order $\sigma = (1, 2, 3, 4)$.

initially permuting the set of points P by some random permutation $\sigma \in S_n$, i.e. processing the points in the order $p_{\sigma(1)}, p_{\sigma(2)}, ..., p_{\sigma(n)}$.

At any time during the algorithm the following information is kept:

① for each simplex we know the points of P, which are contained in it, and
② for each point $p \in P$ we know the simplex containing p.

Expected time complexity In this part we show how to apply the variant of the backwards analysis to derive a bound on the time complexity for the triangulation algorithm. For simplicity we prove the following bound for the 2-dimensional version, even though the analysis can easily be generalized to higher dimensions.

▸ **Theorem 3.1.** *For any fixed d, Algorithm 1 has $O(n \log n)$ expected time complexity.*

The expected running time of the i-th step is dominated by updating the information in Items ① and ②, i.e. rebucketing the remaining points of P which were contained in the deleted triangle τ. These are at most $n - i$ many points, but we show that this number is much less in expectation. For $0 \leqslant j \leqslant i \leqslant n$ and permutation $\sigma \in S_n$ denote by T^i_σ the partial triangulation derived by only processing the first i many points specified by σ. Moreover, let ${}^jA^i_\sigma$ be the union of triangles of T^i_σ, which are incident to point $p_{\sigma(j)}$. Finally, if $i < n$ we write Δ^i_σ for the triangle of T^i_σ, which contains point $p_{\sigma(n)}$.

The main idea for the variant of the backwards analysis is to bound the time complexity needed to process an entire group G_σ by some features of the output derived from a single permutation σ. In the triangulation algorithm case, we are putting the points, which need rebucketing in the i-th step for some permutation $\sigma^{(j)}$ for some $j \in [n]$, in relation with the output triangulation T^i_σ corresponding to permutation σ.

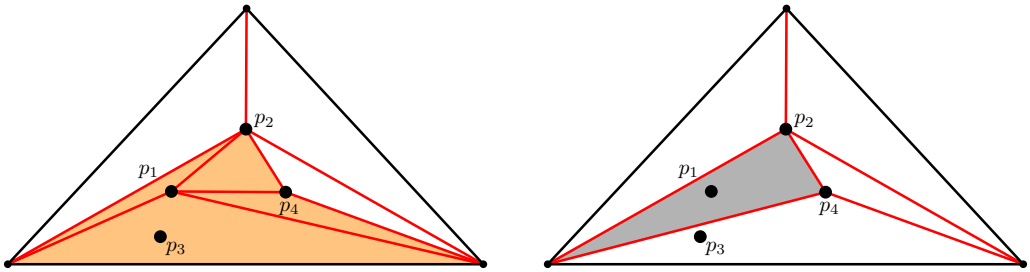

Figure 3 (Left) Partial triangulation T^3_σ for insertion order $\sigma = (2, 1, 4, 3)$ and orange highlighted area ${}^2A^3_\sigma$ and (Right) partial triangulation $T^2_{\sigma^{(2)}}$ for insertion order $\sigma^{(2)} = (2, 4, 3, 1)$ with gray highlighted triangle $\Delta^2_{\sigma^{(2)}}$. Note the subset relation between the orange and gray highlighted sets, which are proven in Lemma 3.2.

▸ **Lemma 3.2.** *For all $1 \leqslant j \leqslant i \leqslant n$ and any permutation $\sigma \in S_n$ it holds: $\Delta^{i-1}_{\sigma^{(j)}} \subset {}^jA^i_\sigma$.*

Proof. The proof works by induction over i. The base case $i = j$ holds because $\Delta^{j-1}_{\sigma^{(j)}} = {}^jA^j_\sigma$. The first difference between $\sigma^{(j)}$ and σ occurs at the j-th position. Thus the triangle in $\mathrm{T}^{j-1}_{\sigma^{(j)}}$, which contains point $p_{\sigma(j)}$, equals the union of three triangles in T^j_σ incident to $p_{\sigma(j)}$.

Let us therefore assume that the induction hypothesis $\Delta^{i-1}_{\sigma^{(j)}} \subset {}^jA^i_\sigma$ holds for some $i \in \{j, j+1, ..., n-1\}$. We want to prove that also $\Delta^i_{\sigma^{(j)}} \subset {}^jA^{i+1}_\sigma$.

If $p_{\sigma(i+1)} \notin {}^jA^i_\sigma$ then we have $\Delta^{i-1}_{\sigma^{(j)}} = \Delta^i_{\sigma^{(j)}}$ and ${}^jA^i_\sigma = {}^jA^{i+1}_\sigma$ and thus the claim $\Delta^i_{\sigma^{(j)}} \subset {}^jA^{i+1}_\sigma$ trivially follows. For the rest of the proof we assume the opposite, i.e. $p_{\sigma(i+1)} \in {}^jA^i_\sigma$. See Fig. 4 for an illustration of the proof. Consider the ray from $p_{\sigma(j)}$ to $p_{\sigma(i+1)}$ and denote q the point of intersection with the boundary of ${}^jA^i_\sigma$. In counterclockwise order around the boundary of ${}^jA^i_\sigma$ we call a (resp. b) the vertex directly after (resp. before) q. Let r_a (resp. r_b) be the ray from $p_{\sigma(i+1)}$ to a (resp. b). Finally, we call W the wedge bounded by r_b and r_a, which does not contain the segment $\overline{p_{\sigma(j)}p_{\sigma(i+1)}}$. Note that none of the points $P \cap {}^jA^i_\sigma$ lies in the interior of W. By construction, we have that ${}^jA^{i+1}_\sigma = {}^jA^i_\sigma \backslash W$. On the other hand, the boundary of the triangle $\Delta^i_{\sigma^{(j)}}$ intersects the segment $\overline{p_{\sigma(j)}p_{\sigma(i+1)}}$, or at least passes through $p_{\sigma(i+1)}$; at the same time it contains the point $p_{\sigma(j)}$ and none of its endpoints lies in W. Therefore, $\Delta^i_{\sigma^{(j)}} \cap W = \emptyset$.

Combining the induction hypothesis $\Delta^{i-1}_{\sigma^{(j)}} \subset {}^jA^i_\sigma$ with the properties $\Delta^i_{\sigma^{(j)}} \subset \Delta^{i-1}_{\sigma^{(j)}}$, $\Delta^i_{\sigma^{(j)}} \cap W = \emptyset$, and ${}^jA^{i+1}_\sigma = {}^jA^i_\sigma \backslash W$, we can indeed verify that also $\Delta^i_{\sigma^{(j)}} \subset {}^jA^{i+1}_\sigma$ holds:

$$\begin{aligned} & \Delta^i_{\sigma^{(j)}} \subset \Delta^{i-1}_{\sigma^{(j)}} \subset {}^jA^i_\sigma \\ \Rightarrow \quad & \Delta^i_{\sigma^{(j)}} \backslash W \subset {}^jA^i_\sigma \backslash W \\ \Rightarrow \quad & \Delta^i_{\sigma^{(j)}} \subset {}^jA^{i+1}_\sigma \end{aligned}$$

◂

▸ **Corollary 3.3.** *In the i-th insertion step of Algorithm 1, each of the $n - i$ remaining points has to be rebucketed for at most 3 permutations within a group G_σ for any $\sigma \in S_i$.*

Proof. Due to Lemma 3.2 we have the property $\Delta^{i-1}_{\sigma^{(j)}} \subset {}^jA^i_\sigma$ holding true for any $j \in [i]$ and permutation $\sigma \in S_i$. The term $\Delta^{i-1}_{\sigma^{(j)}}$ describes exactly the region which has to be re-partitioned in the i-th step of the triangulation algorithm if the permutation $\sigma^{(j)}$ is chosen, i.e. any point of P falling into that region would have to be rebucketed. Thus, the region which needs to be repartitioned for permutation $\sigma^{(j)}$ is a subset of all triangles in T^i_σ, which are incident to point $p_{\sigma(j)}$. For a point $p \in P$ let τ be the triangle of T^i_σ containing p. Then p

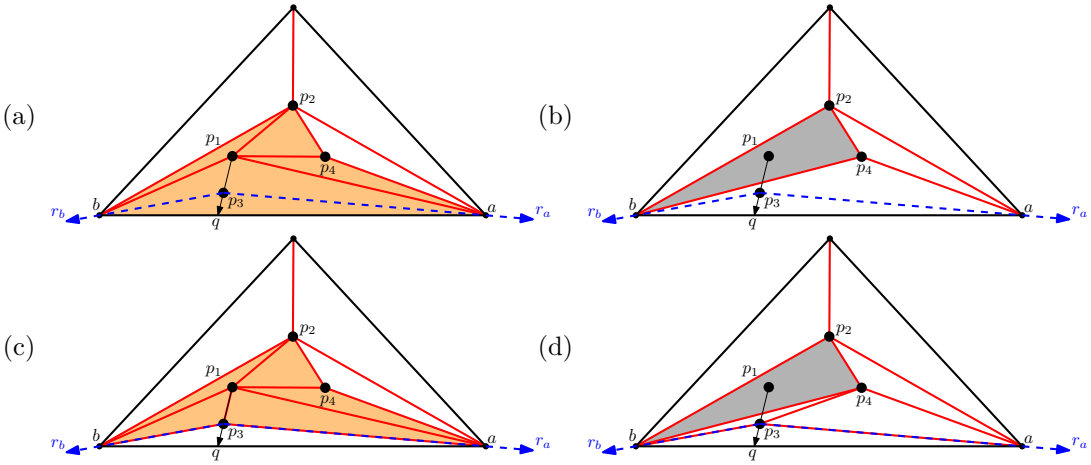

Figure 4 For insertion order $\sigma = (2, 1, 4, 3)$ we show: (a) T^3_σ with highlighted ${}^2A^3_\sigma$ (b) $T^2_{\sigma(2)}$ with highlighted $\Delta^2_{\sigma(2)}$ (c) T^4_σ with highlighted ${}^2A^4_\sigma$ (d) $T^3_{\sigma(2)}$ with highlighted $\Delta^3_{\sigma(2)}$. From $\Delta^2_{\sigma(2)} \subset {}^2A^3_\sigma$ we derive that also $\Delta^3_{\sigma(2)} \subset {}^2A^4_\sigma$.

only needs to be rebucketed for permutations of G_σ, in which one of τ's corners is processed last. Thus p has to be rebucketed at most three times in total for the entire group G_σ; at most once for each permutation in G_σ with one of τ's corners being processed last. ◂

The proof of Theorem 3.1 follows immediately from the previous corollary:

$$E(T(n)) = \frac{\sum_{\sigma \in R_i} T_i(G_\sigma)}{i!} = O\left(\frac{\sum_{\sigma \in R_i} 3(n-i)}{i!}\right) = O\left(\sum_{i=1}^{n} \frac{n}{i}\right) = O(n \log n).$$

4 Conclusion

The first example using this variant of backwards analysis appeared in the context of abstract Voronoi-like diagrams. These were introduced in [5] serving as intermediate structures in a randomized incremental algorithm to perform site-deletion in an abstract Voronoi diagram in expected linear time. These intermediate structures depended on the permutation order of the randomized algorithm, while the final output did not.

The triangulation algorithm, presented in this abstract, is a very simple algorithm, which we mostly used to illustrate this new variant of backwards analysis. We believe that this variant can be of interest to many other problems, when analyzing the expected time complexity of randomized algorithms for order-dependent structures.

References

1 Jean-Daniel Boissonnat and Mariette Yvinec. *Algorithmic Geometry*. Cambridge University Press, New York, NY, USA, 1998.

2 L. Paul Chew. Building Voronoi diagrams for convex polygons in linear expected time. Technical report, Dartmouth College, Hanover, USA, 1990.

3 Kenneth L. Clarkson and Peter W. Shor. Applications of random sampling in computational geometry, II. *Discrete & Computational Geometry*, 4:387–421, 1989.

4 Kolja Junginger and Evanthia Papadopoulou. Deletion in abstract Voronoi diagrams in expected linear time. In *34th International Symposium on Computational Geometry (SoCG)*, volume 99 of *LIPIcs*, pages 50:1–50:14, 2018.

5 Kolja Junginger and Evanthia Papadopoulou. Deletion in abstract Voronoi diagrams in expected linear time and related problems. *Discrete & Computational Geometry*, 69(4):1040–1078, 2023.

6 Vladimir I Levenshtein. On perfect codes in deletion and insertion metric. *Discrete Mathematics and Applications*, 2(3):241–258, 1992.

7 Raimund Seidel. Backwards analysis of randomized geometric algorithms. In *New trends in discrete and computational geometry*, pages 37–67. Springer, 1993.