A Dynamic Algorithm for Well-Spaced Point Sets (Abstract)

Umut A. Acar* Benoît Hudson* Duru Türkoğlu†

The Motivation and the Problem. A set of points is well-spaced if the Voronoi cell of each point has a bounded aspect ratio, i.e., the ratio of the distance to the farthest point in the Voronoi cell divided by the nearest neighbor distance is small. Informally, a set of points is well-spaced if its density varies smoothly. Well-spaced points sets relate strongly to meshing and triangulation for scientific computing: with minimal processing, they lead to quality meshes (e.g., no small angles) in two and higher dimensions. The Voronoi diagram of a well-spaced point set is also immediately useful for the Control Volume Method.

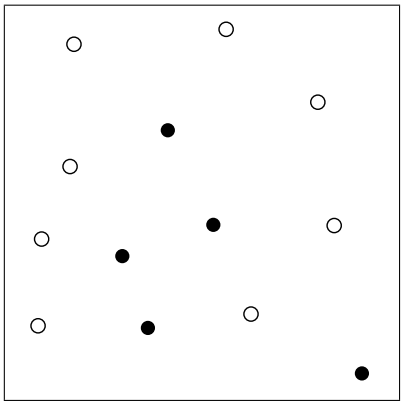

Figure 1: A well-spaced superset. Solid points (•) are input. Empty points (○) are Steiner vertices.

Given a finite set of points N in the d-dimensional unit hypercube $[0,1]^d$, the static well-spaced superset problem is to find a small, well-spaced set $M \supset N$ by inserting so called *Steiner* points. This problem has been studied extensively since the late 1980's, but efficient solutions that can generate outputs no more than a constant factor larger than optimal have been devised only recently. We are interested in the *dynamic* version of the problem. The problem requires maintaining a well-spaced superset (M) while the input (N) changes dynamically due to insertion and deletion of points. When a dynamic change takes place, a dynamic algorithm should efficiently update the output while keeping its size optimal for the new input. There has been relatively little progress on solving the dynamic problem; existing solutions are either not size-optimal or asymptotically no faster than a static algorithm.

Our Contributions. We present a dynamic algorithm that maintains an approximately optimal-sized, well-spaced output and requires worst-case $O(\log \Delta)$ time for an insertion or deletion, which we also prove to be optimal. Here, Δ is the *(geometric) spread* defined as $\frac{1}{\delta}$, where δ is the distance between the closest pair of points in the input. If the spread is polynomially bounded in n, then $O(\log \Delta) = O(\log n)$. We use dynamization to solve the problem. We first design a static algorithm that efficiently constructs an approximately optimal-sized, well-spaced superset of its input. For dynamic updates, the algorithm also constructs a *computation graph*, that represents the operations performed during the execution and the dependences between them. Then, given a modification to the input (i.e., an insertion/deletion), our dynamic algorithm updates the output and the computation graph by identifying the operations that depend on the modification and re-executing them. When re-executing an operation, the update algorithm inserts fresh operations that

*Toyota Technological Institute at Chicago
†University of Chicago

now need to be performed. Similarly, it removes invalid operations that should not be performed. We prove that the update algorithm ensures that the updated computation graph and the output are isomorphic to those that would be obtained by running the static algorithm with the modified input. As a corollary, we conclude that the output is well-spaced and has optimal size.

Proof Idea. The crux of the design and the analysis is to structure the computation in such a way that we can prove that a single modification to the input requires performing a small number of updates. First, we structure the computation into $\Theta(\log \Delta)$ levels (defined by ranks and colors) that the operations in each level depend only on the previous levels. Second, we pick Steiner vertices by making local decisions only, using clipped Voronoi cells. These techniques enable us to process each vertex only once and help isolate and limit the effects of a modification. More specifically, we prove that an insertion/deletion into/from the input causes $O(\log \Delta)$ vertices to become affected. The proof follows from a spacing-and-packing argument. The spacing argument shows that any affected vertex has an empty ball around itself whose radius is proportional to its distance to the vertex inserted or deleted (p). The packing argument shows that there can be only a constant number of affected vertices at each level, consequently, $O(\log \Delta)$ in total. Figure 2 illustrates the proof: each shade corresponds to a rank.

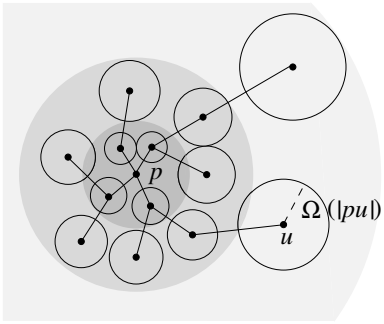

Figure 2: Illustration of the proof.

The Algorithm. We start with a few definitions. Given a vertex set $\mathcal{M}$, the *nearest neighbor distance of v in $\mathcal{M}$*, written $\mathrm{NN}_{\mathcal{M}}(v)$, is the distance from v to the closest other vertex in $\mathcal{M}$. The *Voronoi cell of v in $\mathcal{M}$*, written $\mathrm{Vor}_{\mathcal{M}}(v)$, consists of points $x \in [0,1]^d$ such that for all $u \in \mathcal{M}$, $|vx| \leq |ux|$. The *outradius* of $\mathrm{Vor}_{\mathcal{M}}(v)$ is the distance from v to the farthest point in $\mathrm{Vor}_{\mathcal{M}}(v)$ and the *aspect ratio* is outradius divided by $\mathrm{NN}_{\mathcal{M}}(v)$. For a given quality criterion $\rho > 1$, we say that a vertex v is *ρ-well-spaced* if the aspect ratio of its Voronoi cell is bounded by ρ and $\mathcal{M}$ is *ρ-well-spaced* if every point in $\mathcal{M}$ is ρ-well-spaced. For any constant $\beta > \rho$, we define the *β-clipped Voronoi cell*, written $\mathrm{Vor}^{\beta}_{\mathcal{M}}(v)$, as the intersection of $\mathrm{Vor}_{\mathcal{M}}(v)$ with the ball of radius $\beta\,\mathrm{NN}_{\mathcal{M}}(v)$ centered at v. Note that $\mathrm{Vor}^{\beta}_{\mathcal{M}}(v) \setminus \mathrm{Vor}^{\rho}_{\mathcal{M}}(v)$ is empty if and only if v is ρ-well-spaced.

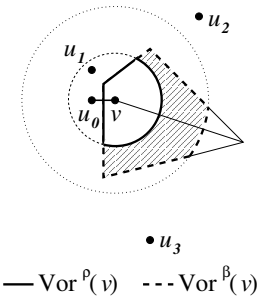

Figure 3: Nearest neighbor, outradius, ρ and β-clipped Voronoi cells.

Our static algorithm structures the computation into $\Theta(\log \Delta)$ ranks based on the nearest neighbor distances of the vertices and performs a single pass over the vertices ordered by their ranks. This technique requires selecting the Steiner vertices without affecting a previously processed vertex. Therefore, while processing a vertex $v \in \mathcal{M}$, we insert Steiner vertices only within the Voronoi cell of v, $\mathrm{Vor}_{\mathcal{M}}(v)$, but still far from v: at least at distance $\rho\,\mathrm{NN}_{\mathcal{M}}(v)$. Moreover, in order to support efficient updates, we select Steiner vertices by making local decisions only. More specifically, we select Steiner vertices within the β-clipped Voronoi cell of v; from $\mathrm{Vor}^{\beta}_{\mathcal{M}}(v) \setminus \mathrm{Vor}^{\rho}_{\mathcal{M}}(v)$ (Figure 3 shows this region). Even if we select Steiner vertices locally, these local decisions may combine and create long dependence chains which would require larger global restructuring during dynamic updates. To address this problem we partition the work in each rank into constantly many color classes. At each rank, we process the vertices in monotonically increasing order of their colors. This ensures independence: the Steiner vertices we insert while processing a vertex do not affect the decisions we make when processing other vertices of the same color. These techniques enable us to prove that dependence chains are short, of length $O(\log \Delta)$.