An Efficient, Practical Algorithm and Implementation for Computing Multiplicatively Weighted Voronoi Diagrams

Martin Held[1] Stefan de Lorenzo[1]

[1] University of Salzburg, Department of Computer Science

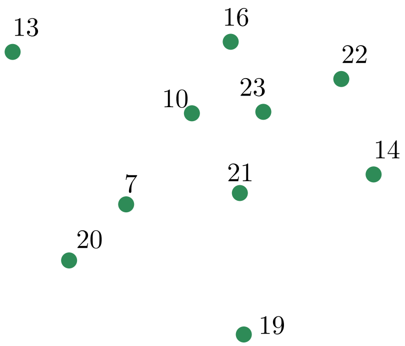
September 7, 2020

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Problem Specification

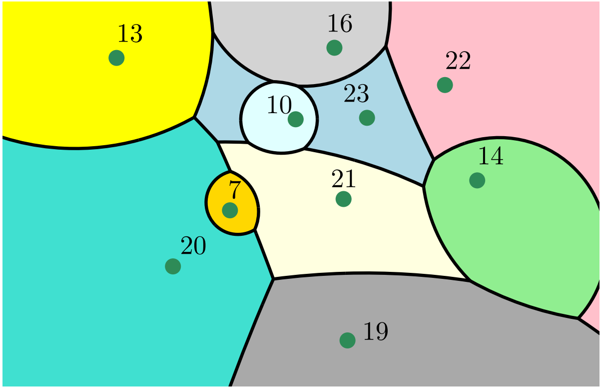
Problem

Given: A set S of n input points in the plane, where every $s \in S$ is associated with a weight $w(s) > 0$.

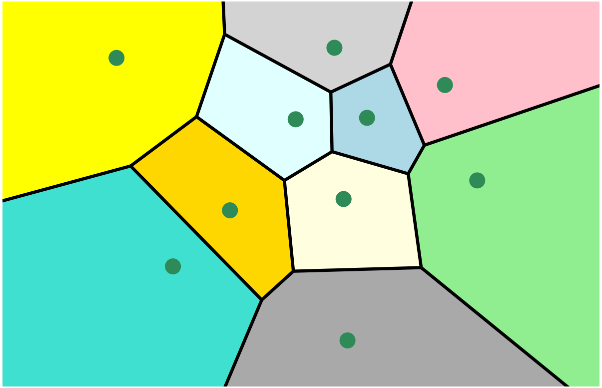


Problem

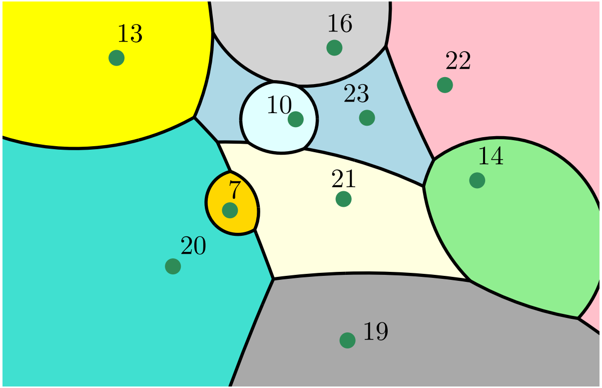
Given: A set S of n input points in the plane, where every $s \in S$ is associated with a weight $w(s) > 0$.
Compute: The ***multiplicatively weighted Voronoi diagram (MWVD)*** $\mathcal{VD}_w(S)$ of S.



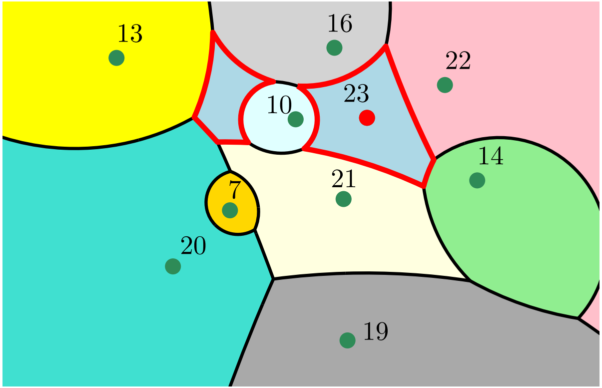
- Every Voronoi edge is given by a straight-line segment or a ray.
- The Voronoi regions are convex.
- The (unweighted) Voronoi diagram has a linear combinatorial complexity in the worst case.



- The Voronoi edges are formed by straight-line segments, rays, and circular arcs.

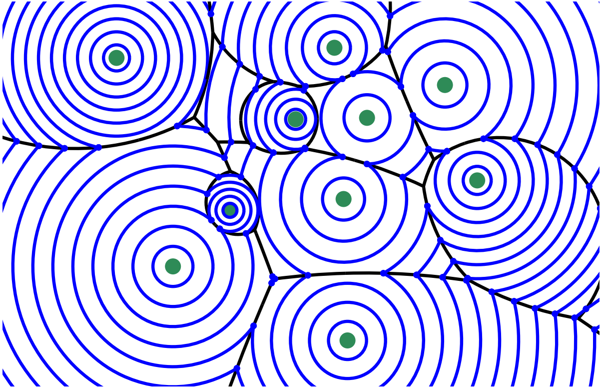


- The Voronoi edges are formed by straight-line segments, rays, and circular arcs.
- The Voronoi regions are (possibly) disconnected.
- The MWVD has a quadratic combinatorial complexity in the worst case.

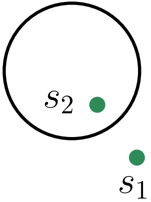


A Wavefront-Based Strategy

- We present a wavefront-based approach for computing MWVDs.
- The ***wavefront*** covers an increasing portion of the plane over time.
- It consists of ***wavefront arcs*** and ***wavefront vertices***.
- Whenever a wavefront arc appears or disappears, a new Voronoi node is discovered.

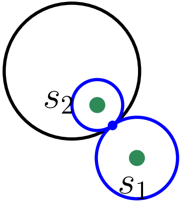


- Every site is associated with an ***offset circle***.



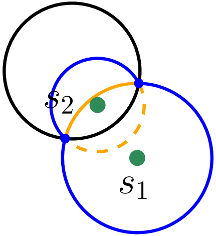
Offset Circles

- Every site is associated with an ***offset circle***.
- Two ***moving vertices*** trace out the bisector as time progresses.



Offset Circles

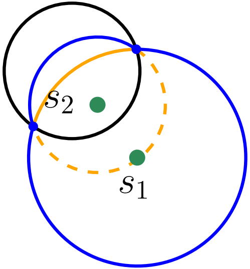
- Every site is associated with an ***offset circle***.
- Two ***moving vertices*** trace out the bisector as time progresses.
- ***Inactive arcs*** along the offset circles are eliminated.
- The ***active arcs*** are stored in sorted angular order.



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

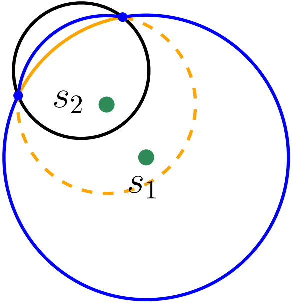
Offset Circles

- Every site is associated with an ***offset circle***.
- Two ***moving vertices*** trace out the bisector as time progresses.
- ***Inactive arcs*** along the offset circles are eliminated.
- The ***active arcs*** are stored in sorted angular order.



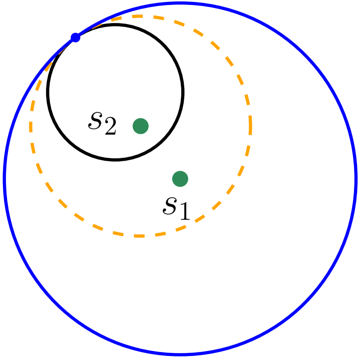
Offset Circles

- Every site is associated with an ***offset circle***.
- Two ***moving vertices*** trace out the bisector as time progresses.
- ***Inactive arcs*** along the offset circles are eliminated.
- The ***active arcs*** are stored in sorted angular order.



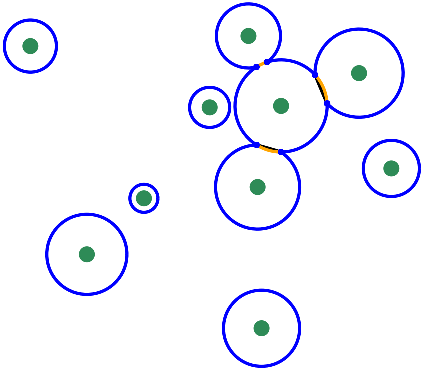
Offset Circles

- Every site is associated with an ***offset circle***.
- Two ***moving vertices*** trace out the bisector as time progresses.
- ***Inactive arcs*** along the offset circles are eliminated.
- The ***active arcs*** are stored in sorted angular order.



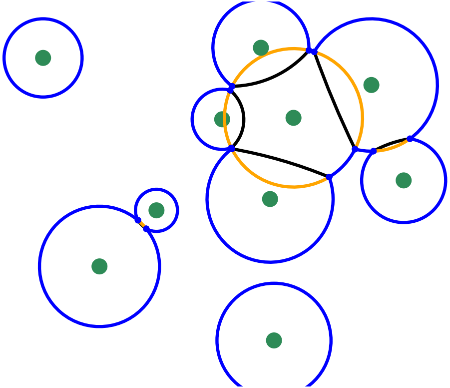
Event Handling

- ***Collision*** and ***domination events*** mark the initial and last contact of two offset circles.
- ***Arc events*** happen whenever active arcs appear or disappear.
- These events are stored in a priority queue $\mathcal{Q}$.
- The angular order of active arcs only changes at events.

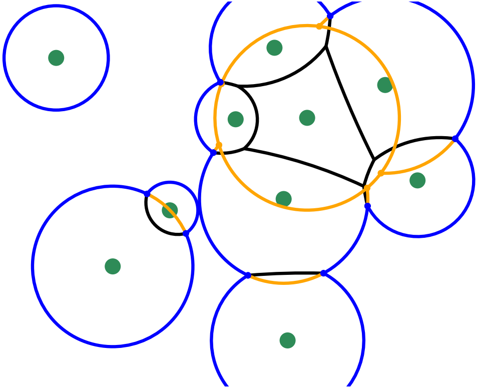


Event Handling

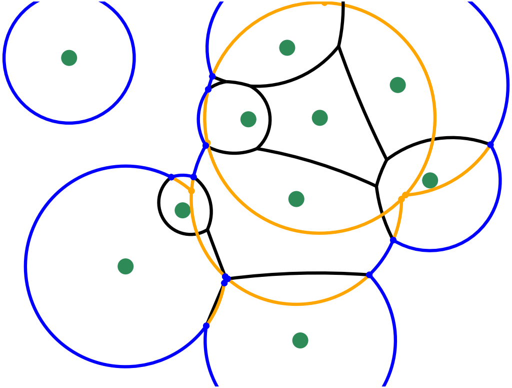
- ***Collision*** and ***domination events*** mark the initial and last contact of two offset circles.
- ***Arc events*** happen whenever active arcs appear or disappear.
- These events are stored in a priority queue $\mathcal{Q}$.
- The angular order of active arcs only changes at events.



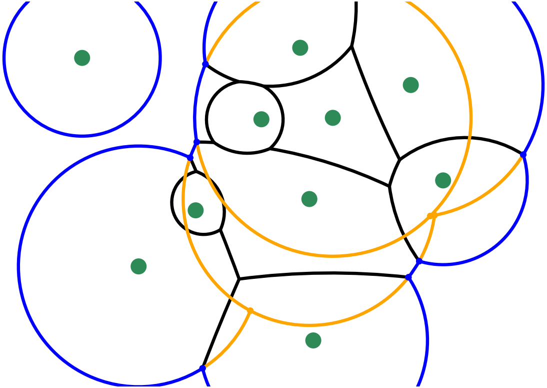
- ***Collision*** and ***domination events*** mark the initial and last contact of two offset circles.
- ***Arc events*** happen whenever active arcs appear or disappear.
- These events are stored in a priority queue $\mathcal{Q}$.
- The angular order of active arcs only changes at events.



- ***Collision*** and ***domination events*** mark the initial and last contact of two offset circles.
- ***Arc events*** happen whenever active arcs appear or disappear.
- These events are stored in a priority queue $\mathcal{Q}$.
- The angular order of active arcs only changes at events.

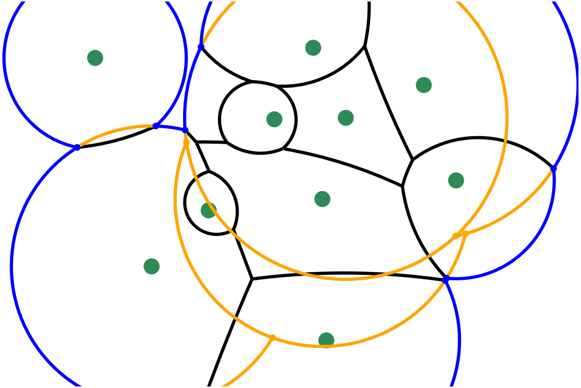


- ***Collision*** and ***domination events*** mark the initial and last contact of two offset circles.
- ***Arc events*** happen whenever active arcs appear or disappear.
- These events are stored in a priority queue $\mathcal{Q}$.
- The angular order of active arcs only changes at events.

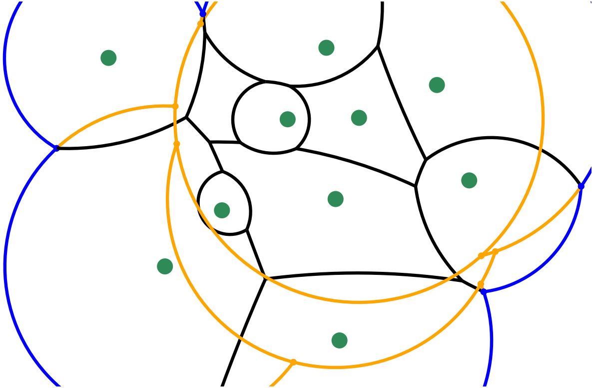


Event Handling

- ***Collision*** and ***domination events*** mark the initial and last contact of two offset circles.
- ***Arc events*** happen whenever active arcs appear or disappear.
- These events are stored in a priority queue $\mathcal{Q}$.
- The angular order of active arcs only changes at events.

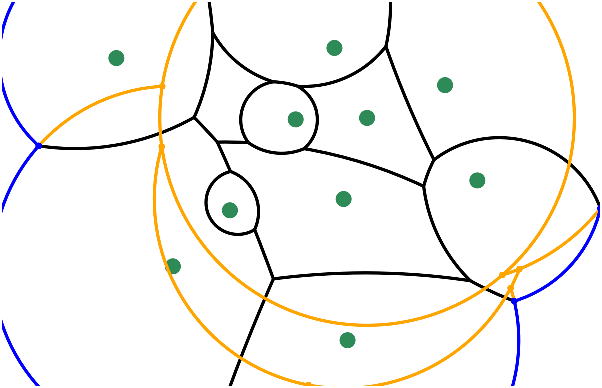


- ***Collision*** and ***domination events*** mark the initial and last contact of two offset circles.
- ***Arc events*** happen whenever active arcs appear or disappear.
- These events are stored in a priority queue $\mathcal{Q}$.
- The angular order of active arcs only changes at events.



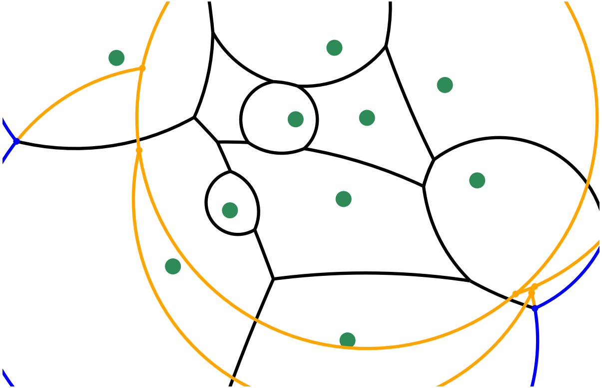
Event Handling

- ***Collision*** and ***domination events*** mark the initial and last contact of two offset circles.
- ***Arc events*** happen whenever active arcs appear or disappear.
- These events are stored in a priority queue $\mathcal{Q}$.
- The angular order of active arcs only changes at events.

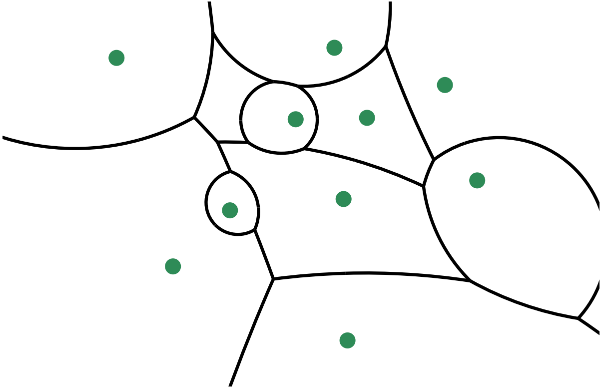


Event Handling

- ***Collision*** and ***domination events*** mark the initial and last contact of two offset circles.
- ***Arc events*** happen whenever active arcs appear or disappear.
- These events are stored in a priority queue $\mathcal{Q}$.
- The angular order of active arcs only changes at events.



- ***Collision*** and ***domination events*** mark the initial and last contact of two offset circles.
- ***Arc events*** happen whenever active arcs appear or disappear.
- These events are stored in a priority queue $\mathcal{Q}$.
- The angular order of active arcs only changes at events.

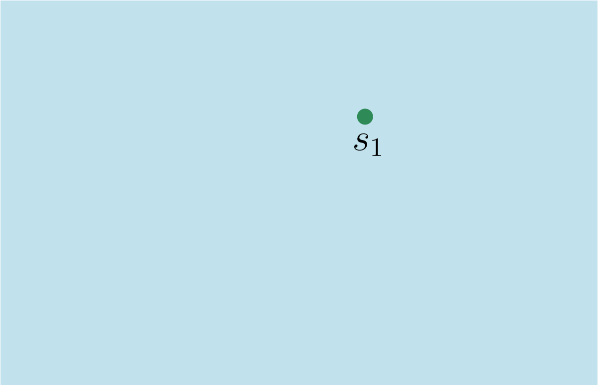


Event Handling

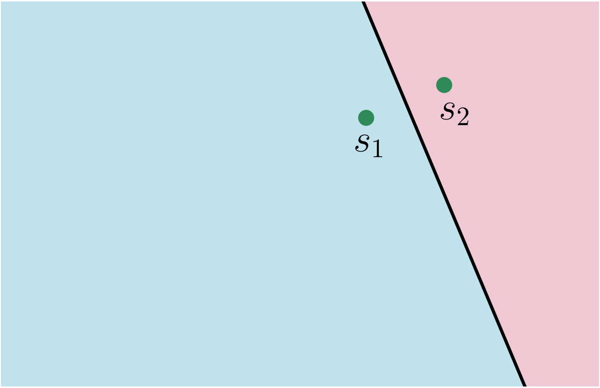
- All topological changes of the wavefront are properly detected.
- A quadratic number of collision events are computed in any case.
- A moving vertex can be charged with a constant number of arc events.
- In the worst case $\mathcal{O}(n^2)$ arc events take place.
- All events can be handled in $\mathcal{O}(\log n)$ time.
- Therefore, the algorithm's runtime equals $\mathcal{O}(n^2 \log n)$ in the worst case.

Reducing the Number of Collisions

- A vast number of collisions are invalid for general input.
- The calculation of all possible collision requires a high computational effort.
- Invalid collision are filtered in an additional preprocessing step.
- The average case behavior of the algorithm is improved by using an ***overlay arrangement***.

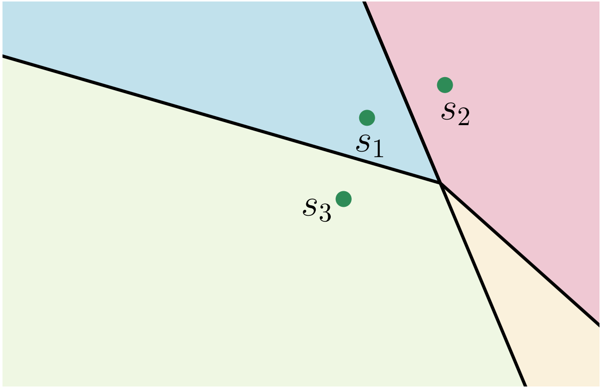


- A vast number of collisions are invalid for general input.
- The calculation of all possible collision requires a high computational effort.
- Invalid collision are filtered in an additional preprocessing step.
- The average case behavior of the algorithm is improved by using an ***overlay arrangement***.

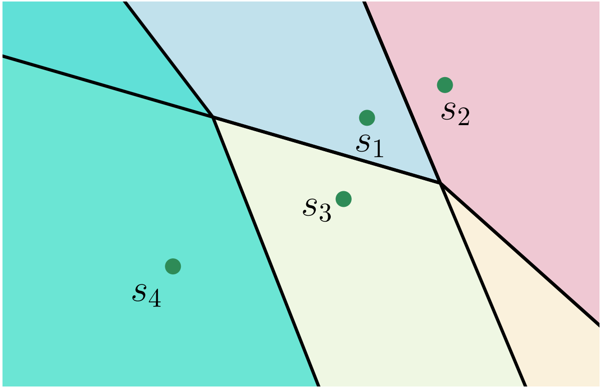


Reducing the Number of Collisions

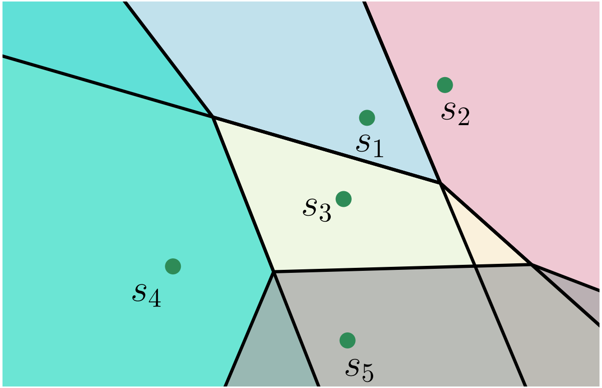
- A vast number of collisions are invalid for general input.
- The calculation of all possible collision requires a high computational effort.
- Invalid collision are filtered in an additional preprocessing step.
- The average case behavior of the algorithm is improved by using an ***overlay arrangement***.



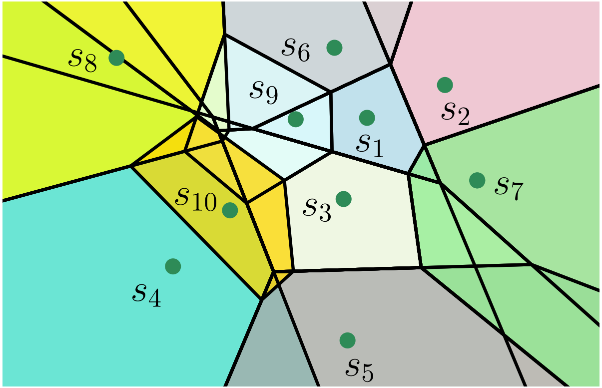
- A vast number of collisions are invalid for general input.
- The calculation of all possible collision requires a high computational effort.
- Invalid collision are filtered in an additional preprocessing step.
- The average case behavior of the algorithm is improved by using an ***overlay arrangement***.



- A vast number of collisions are invalid for general input.
- The calculation of all possible collision requires a high computational effort.
- Invalid collision are filtered in an additional preprocessing step.
- The average case behavior of the algorithm is improved by using an ***overlay arrangement***.

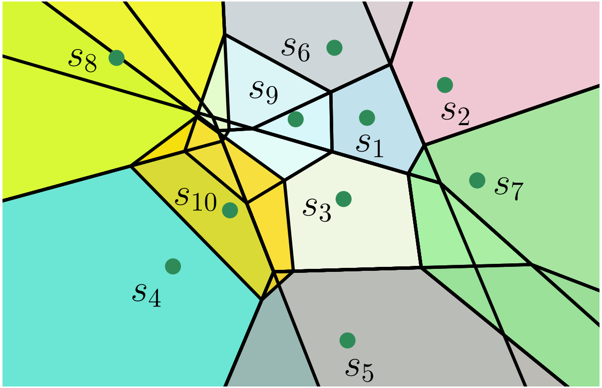


- A vast number of collisions are invalid for general input.
- The calculation of all possible collision requires a high computational effort.
- Invalid collision are filtered in an additional preprocessing step.
- The average case behavior of the algorithm is improved by using an ***overlay arrangement***.



Candidate Set

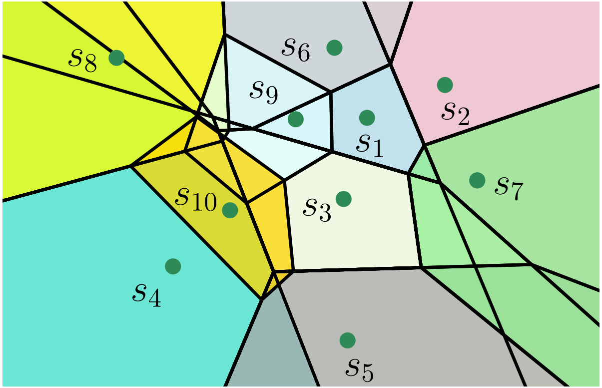
The ***candidate set*** for a weighted nearest neighbor of $q \in \mathbb{R}^2$ consists of all sites $s \in S$ such that all other sites in S either have a smaller weight or a larger Euclidean distance to q.



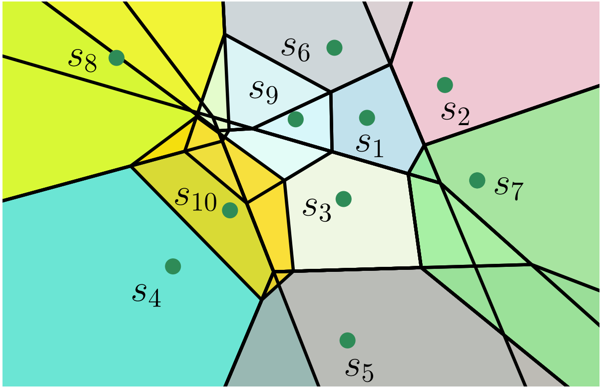
Candidate Set

The ***candidate set*** for a weighted nearest neighbor of $q \in \mathbb{R}^2$ consists of all sites $s \in S$ such that all other sites in S either have a smaller weight or a larger Euclidean distance to q.

- Only sites within the same candidate set may collide.



- A candidate set contains $\mathcal{O}(\log n)$ many sites in the expected case [HPR15].
- The expected complexity of overlay arrangement is bounded by $\mathcal{O}(n \log n)$ [KRS11].
- Thus, we may expect to compute $\mathcal{O}(n \log^3 n)$ many collisions.
- Our improved strategy computes $\mathcal{VD}_w(S)$ in expected $O(n \log^4 n)$ time.

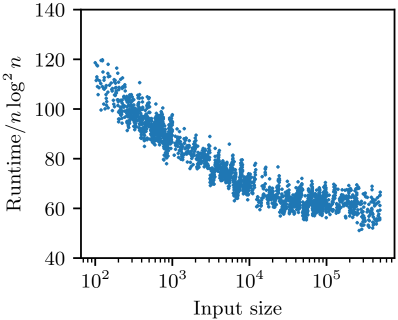


Implementation

- The implementation is based on the Computational Geometry Algorithms Library (CGAL).
- Our code is available on GitHub under `https://github.com/cgalab/wevo`.
- It can be freely used under the GNU General Public License 3.

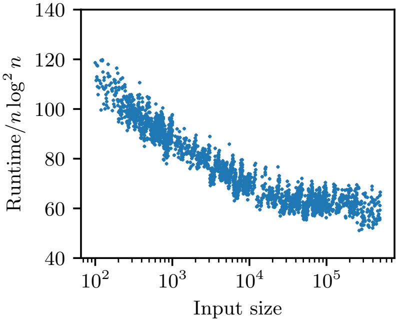
Experimental Evaluation: Runtime

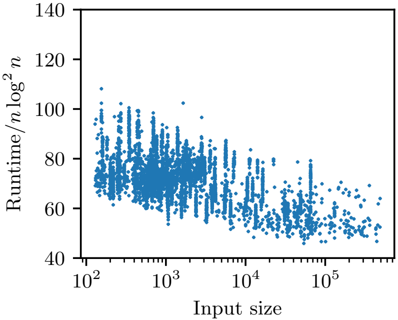
- We tested our strategy on over 8000 different inputs ranging from 256 vertices to 500 000 vertices.
- All tests were carried out on an Intel Core i9-7900X processor clocked at 3.3 GHz.
- For all of these inputs, the weights were chosen uniformly at random.
- The point locations were either randomly chosen ...



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

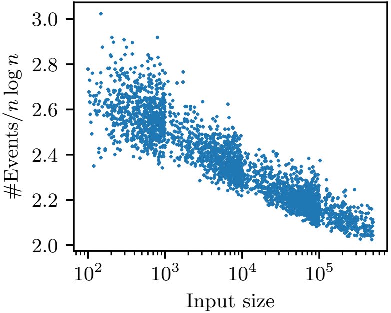
- We tested our strategy on over 8000 different inputs ranging from 256 vertices to 500 000 vertices.
- All tests were carried out on an Intel Core i9-7900X processor clocked at 3.3 GHz.
- For all of these inputs, the weights were chosen uniformly at random.
- The point locations were either randomly chosen or derived from the ***Salzburg Database of Polygonal Data*** [EHJ+20].





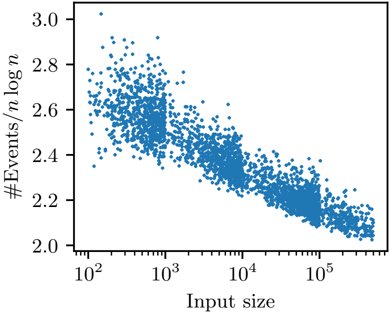
Experimental Evaluation: Number of Events

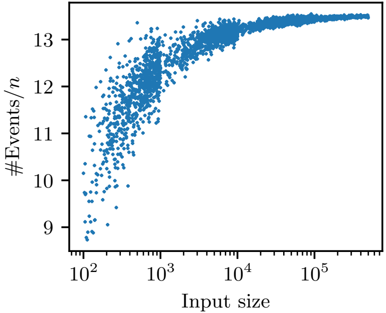
- We tracked the number of collision ...



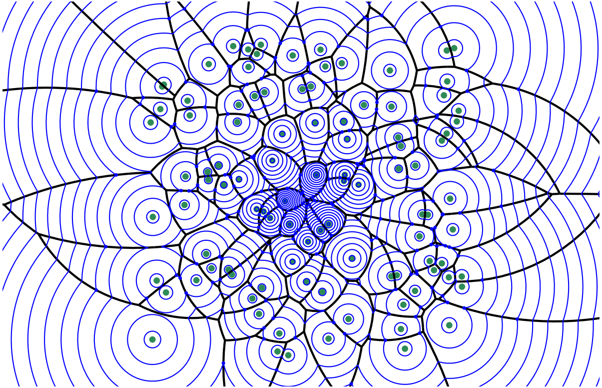
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

- We tracked the number of collision and arc events that occurred during our test runs.
- The number of arc events forms an upper bound on the number of Voronoi nodes.
- Random weights seem to result in a linear combinatorial complexity of the final MWVD.

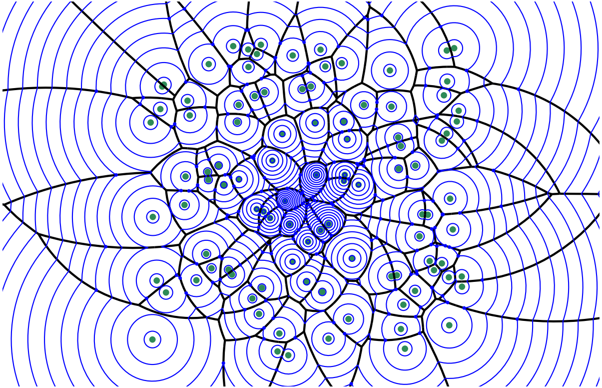




- The actual geometric distribution of the sites does not have a significant impact on the runtime.
- Our expected-case bounds only apply for inputs whose weights are chosen randomly.



- The actual geometric distribution of the sites does not have a significant impact on the runtime.
- Our expected-case bounds only apply for inputs whose weights are chosen randomly.
- How much do our experimental results depend on the randomness of the weights?



Experimental Evaluation: Non-Random Weights

- We sampled points uniformly within a square with side-length $\sqrt{2}$.

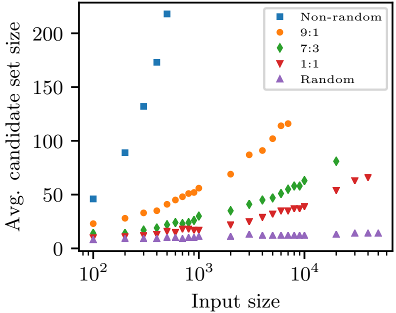
Experimental Evaluation: Non-Random Weights

- We sampled points uniformly within a square with side-length $\sqrt{2}$.
- Let $d(s)$ be the distance of the site $s \in S$ from the center of the square, and let $r(s)$ be a number uniformly distributed within the interval $[0, 1]$.

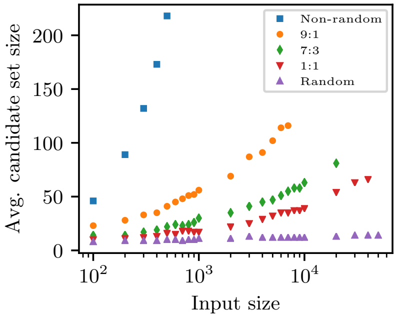
Experimental Evaluation: Non-Random Weights

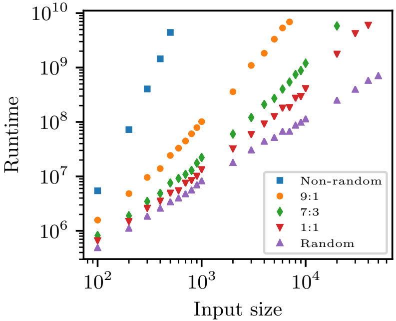
- We sampled points uniformly within a square with side-length $\sqrt{2}$.
- Let $d(s)$ be the distance of the site $s \in S$ from the center of the square, and let $r(s)$ be a number uniformly distributed within the interval $[0, 1]$.
- We assign ${}^{\alpha \cdot d(s) + \beta \cdot r(s)}/_{(\alpha+\beta)}$ as weight to s, where $\alpha > 0$ and $\beta > 0$ are two arbitrary but fixed numbers for all $s \in S$.

- We sampled points uniformly within a square with side-length $\sqrt{2}$.
- Let $d(s)$ be the distance of the site $s \in S$ from the center of the square, and let $r(s)$ be a number uniformly distributed within the interval $[0, 1]$.
- We assign ${}^{\alpha \cdot d(s) + \beta \cdot r(s)}/_{(\alpha+\beta)}$ as weight to s, where $\alpha > 0$ and $\beta > 0$ are two arbitrary but fixed numbers for all $s \in S$.



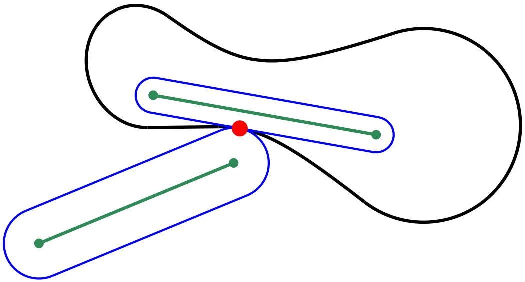
- We sampled points uniformly within a square with side-length $\sqrt{2}$.
- Let $d(s)$ be the distance of the site $s \in S$ from the center of the square, and let $r(s)$ be a number uniformly distributed within the interval $[0, 1]$.
- We assign ${}^{\alpha \cdot d(s) + \beta \cdot r(s)}/_{(\alpha+\beta)}$ as weight to s, where $\alpha > 0$ and $\beta > 0$ are two arbitrary but fixed numbers for all $s \in S$.





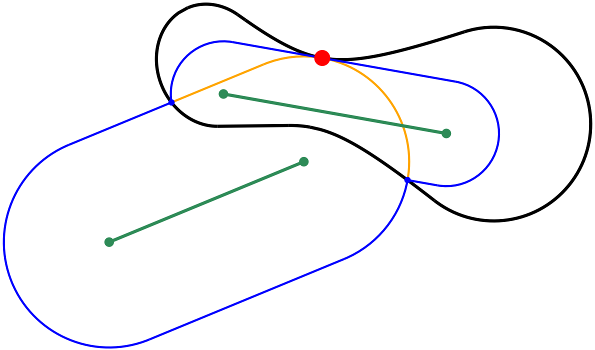
Extension the Straight-Line Segments

- The strategy can be easily extended to also handle weighted straight-line segments.
- Our notion of a "collision" needs to be refined.
- Thus, we distinguish between ***non-piercing*** ...

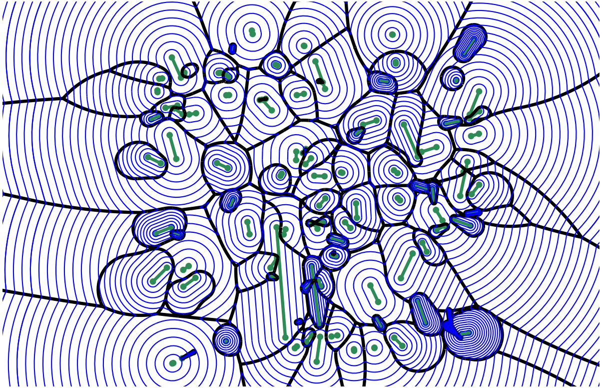


Extension the Straight-Line Segments

- The strategy can be easily extended to also handle weighted straight-line segments.
- Our notion of a “collision” needs to be refined.
- Thus, we distinguish between ***non-piercing*** and ***piercing collision events***.
- Whenever a piercing collision event occurs, a second pair of moving vertices appears.



- The strategy can be easily extended to also handle weighted straight-line segments.
- Our notion of a “collision” needs to be refined.
- Thus, we distinguish between ***non-piercing*** and ***piercing collision events***.
- Whenever a piercing collision event occurs, a second pair of moving vertices appears.



Additional Extensions

- Our (basic) algorithm is also able to deal with ***additive weights***.
- Every offset circle that is associated with an additive weight gets a head-start.

Additional Extensions

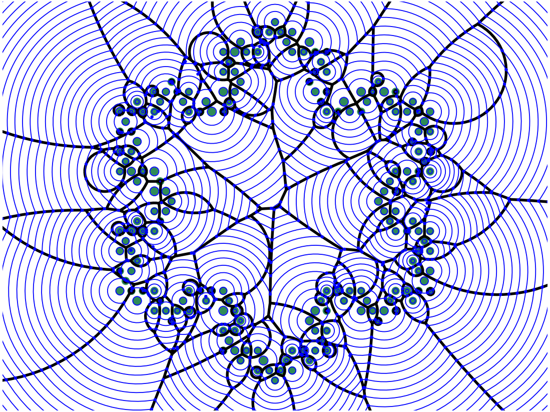
- Our (basic) algorithm is also able to deal with ***additive weights***.
- Every offset circle that is associated with an additive weight gets a head-start.
- Our wavefront-based algorithm implies a simple strategy for computing one-dimensional MWVDs.

Additional Extensions

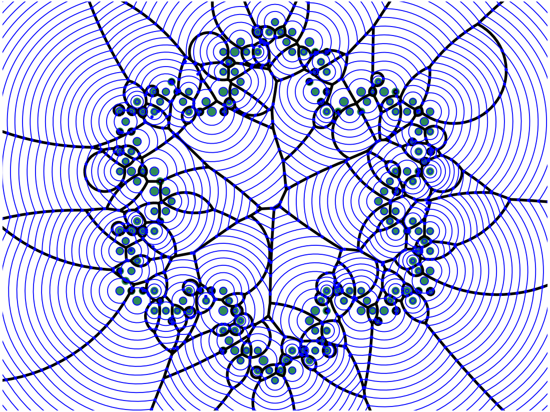
- Our (basic) algorithm is also able to deal with ***additive weights***.
- Every offset circle that is associated with an additive weight gets a head-start.
- Our wavefront-based algorithm implies a simple strategy for computing one-dimensional MWVDs.
- Thus, the one-dimensional MWVD can be computed in worst-case optimal $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space.

Discussion

- We propose a fast, practical strategy to compute MWVDs.
- The expected runtime is improved by using an overlay arrangement.
- We provide a robust implementation using exact arithmetic.



- We propose a fast, practical strategy to compute MWVDs.
- The expected runtime is improved by using an overlay arrangement.
- We provide a robust implementation using exact arithmetic.



Thank you for your attention!

References I

Günther Eder, Martin Held, Steinþór Jasonarson, Philipp Mayer, and Peter Palfrader.
Salzburg Database of Polygonal Data: Polygons and Their Generators.
Data in Brief, 31:105984, August 2020.

Sariel Har-Peled and Benjamin Raichel.
On the Complexity of Randomly Weighted Multiplicative Voronoi Diagrams.
Discrete & Computational Geometry, 53(3):547–568, 2015.

Haim Kaplan, Edgar Ramos, and Micha Sharir.
The Overlay of Minimization Diagrams in a Randomized Incremental Construction.
Discrete & Computational Geometry, 45(3):371–382, 2011.