Also publ. as paper in: Proceedings of the 5th Annual International Symposium on Voronoi Diagrams in Science and Engineering, 22-28 September, 2008, Kiev, Ukraine

CAPACITY-CONSTRAINED VORONOI DIAGRAMS IN FINITE SPACES

MICHAEL BALZER AND DANIEL HECK

ABSTRACT. A Voronoi diagram of a set of n sites partitions a finite set of m points into regions of different areas, called the capacities of the sites. In this paper we are interested in Voronoi diagrams in which the capacities are given as constraints. We compute such capacity-constrained Voronoi diagrams in finite spaces by starting with an arbitrary partition that fulfills the capacity constraint without representing a valid Voronoi diagram. We then iteratively swap the assignment of points to sites guided by an energy minimization that is related to the distance function of the Voronoi diagram. This optimization converges towards a valid Voronoi diagram that fulfills the capacity constraint due to the restriction to swap operations. The straightforward structure of our algorithm makes it easy to implement and flexible. The computational time complexity for each iteration step is $O(n^2 + nm \log \frac{m}{n})$. We provide examples for extensions of our algorithm such as applying different energy functions, or generating Voronoi diagrams with centroidal and/or void regions.

1. INTRODUCTION

A Voronoi diagram of a set of n sites partitions the Euclidean d-space into n regions. The region of each site consists of all points that are closer to this site than to any other site. The areas of the regions are implicitly defined by the distances between neighboring sites. It is possible to influence the areas of the regions by assigning an additional set of weights to the sites, and varying the distance function. However, the actual areas of the regions still depend on the arrangement of the sites, and the relations between the weights. See [1, 6] for an overview of Voronoi diagrams and their properties.

In this paper, we are interested in Voronoi diagrams in which the region areas are given as constraints, which means that the resulting areas of the regions are predefined. We call this predefined region area the *capacity* of the site. Due to the missing direct relation between the weights of the sites and the resulting areas of the regions, generating Voronoi diagrams under this constraint is a non-trivial task. Aurenhammer et al. [2] presented an approach for generating capacity-constrained power diagrams for a set of given sites in finite and continues spaces, and proved their equivalence to similarly constrained least-squares assignments and Minkowski's theorem for convex polytopes, respectively. They iteratively generate the Voronoi diagram and adapt the weights of the sites according to the violation of the capacity constraints. The same approach can also be used for variable site locations and other distance functions, like additively and multiplicatively weighted Voronoi diagrams [3, 7].

2000 *Mathematics Subject Classification.* Primary 05B45; Secondary 93E24.

Key words and phrases. Voronoi diagram, capacity constraint, finite space, power diagram, least-squares, centroidal Voronoi diagram.

In contrast, our approach for finite spaces starts with an arbitrary partition of the finite space under consideration. This initial partition must fulfill the capacity constraint for all regions without exception, but does not have to represent a valid Voronoi diagram. We then iteratively swap the assignment of points to sites guided by an energy minimization. The utilized energy function is directly related to the distance function of the Voronoi diagram. This optimization converges towards a valid Voronoi diagram that still fulfills the initial capacity constraint due to the restriction to swap operations during the optimization. The main advantage of our approach over existing methods is its flexibility: It can be used with different finite spaces, different energy functions for achieving different Voronoi diagrams, and additional application-specific constraints, such as movable sites, void regions, or density functions. Furthermore, our approach is easy to implement, and possesses a competitive computational complexity.

The remainder of the paper is structured as follows: Section 2 discusses the necessary theoretic background, introduces our algorithm for computing capacity-constrained power diagrams in finite spaces, and discusses its properties. Section 3 presents extensions to our algorithm to generate capacity-constrained Voronoi diagrams with other distance functions and additional constraints. Finally, a conclusion and outlook to future work are given in Section 4.

2. METHOD

2.1. **Background.** A set S of n sites in Euclidean d-space $\mathbb{E}^d$ induces a partition of $\mathbb{E}^d$ into n regions. The region $reg(s_i)$ of a site $s_i \in S$ consists of all points x that are closer to s_i than to any other site $s_j \in S$, $i \neq j$. This partition is known as the Voronoi diagram of S in $\mathbb{E}^d$. For any finite set X of m points, which determines a *finite space* in $\mathbb{E}^d$, the Voronoi diagram of S induces a partition of X into subsets. It defines an *assignment* $A : X \to S$, given by $A(x) = s \Longleftrightarrow x \in reg(s)$, $s \in S$. Note that points that are equally close to more than one site are not covered by this definition. Rather, the function A assigns such point to an arbitrary but fixed closest site. The number of points assigned to a particular site s, $|reg(s)|$, is called the *capacity* $c(s)$, whereby the sum of the capacities of all sites is $m = |X|$.

The capacity of a site depends on the arrangement of all sites, their pairwise Euclidean distances, and the underlying distance function $\delta(x,s)$ that governs the assignment of points to sites. Ordinary Voronoi diagrams use the Euclidean distance as their distance function. Weighted Voronoi diagrams attach an additional set $W = \{w(s)|s \in S\}$ of real numbers, called *weights*, to the sites, and vary the distance function in such a way that the weights influence the distances between points and sites. This means that if the weight of a site is increased (or decreased), their resulting region is expanded (or shrunken), for example. Thus, the capacities of the sites can be controlled by adjusting their weights. Again, we obtain an assignment function $A_w : X \to S$ which now depends on the set of weights. Unfortunately, there exists no closed-form relation between the weight of a site and its resulting capacity.

Consequently, if we want determine the assignment A_w that partitions a finite set X of points according to a finite set S of sites in $\mathbb{E}^d$ in such a way that the set $C = \{c(s)|s \in S\}$ of capacities consists of given integer values $c \geq 0$, with $\sum C = |X|$, we have to find the set W of weights. For the power diagram, which uses the distance

function

$$\delta_{pow}(x,s) = \|x-s\|^2 - w(s), \tag{1}$$

Aurenhammer et al. [2] show that there always exists such a set of weights that fulfills a given capacity constraint C. Furthermore, they demonstrated the equivalence of such a capacity-constrained power diagram to a similarly constrained Euclidean least-squares assignment. In this process, he showed that the power diagram minimizes the expression

$$\sum_{x\in X} \delta_{pow}(x, A_w(x)) = \sum_{x\in X} \|x - A_w(x)\|^2 - \sum_{x\in X} w(A_w(x)) \tag{2}$$

over all possible assignments $A_w : X \to S$, regardless of the capacity constraint C. Here, the last sum is a constant for all assignments, and can therefore be omitted. Thus, a capacity-constrained power diagram can be generated by performing a least-squares minimization.

The algorithmic approach by Aurenhammer et al. to compute a capacity-constrained power diagram, for the finite and continuous case, is to iteratively adjust the set of weights W based on emerging violations of the capacity constraint C. "As a byproduct" [2] this algorithm also determines the desired assignment A_w. Similar iterative capacity-constrained approaches were used by Balzer and Deussen [3] to generate power diagrams and additively weighted Voronoi diagrams for the continuous case, and by Reitsma et al. [7] to generate multiplicatively weighted Voronoi diagrams for the finite case.

Our approach tackles the problem from the opposite direction: We ignore the weights but maintain the capacity constraint from the very beginning. As shown in Equation 2, the power diagram is equivalent to a least-squares minimization for which the set of weights is ignored. Hence, we obtain a power diagram of a finite site set S for the finite point set X that fulfills a given capacity constraint C by performing a least-squares minimization on an arbitrary assignment $A : X \to S$ with the same capacity constraint C. In other words, the least-squares minimization of any capacity-constrained assignment results in a valid power diagram with the same capacity constraint, if this capacity constraint is preserved in all minimization steps. Of course, we do not obtain the weights for the sites of the power diagram, but this does not affect the correctness of the result. The following sections will describe and discuss our approach in detail.

2.2. **Algorithm.** The goal of our algorithm is to obtain a partition of a finite set X of m points into n subsets according to a set S of n sites with capacity constraints C, where $\sum C = m$. This partition represents a power diagram of S for the point set X. We are not interested in the actual weights of the power diagram. The basic concept of our algorithm is that we start with an arbitrary assignment $A : X \to S$ that fulfills the capacity constraint C, and then perform an iterative least-squares minimization. Throughout this minimization, we preserve the capacity constraint C by exclusively swapping the assignment of two points belonging to different sites. Finally, the minimization converges in a stable state that represents a valid power diagram. The time complexity of the algorithm is $O(n^2 + nm\log\frac{m}{n})$ for each iteration step, and its memory complexity is $O(n+m)$. Our method is described in detail in the next paragraphs, and furthermore outlined in Algorithm 1.

We start with an arbitrary initialization of the assignment $A : X \rightarrow S$ in which the number of points $|reg(s_i)|$ assigned to each site $s_i \in S$ equals the desired capacity $c(s_i)$. Such initial assignment can be generated by randomly assigning points in X to sites in S as long as their capacity is not exceeded. Note that the sum of all capacities in C must be equal to the number of points in X. The initial assignment is most likely not equivalent to the corresponding power diagram of the sites in S. Rather, we have to perform a least-squares minimization, which is directly related to the distance function of the power diagram as shown in Equation 2.

The basic operation during the energy minimization is to swap the assignment of two points x_i, x_j belonging to different sites s_i, s_j. The restriction to swap operations guarantees that the capacity constraint for each site is preserved throughout the energy minimization. Of course, finding the minimal series of swaps that yields the global minimum of the energy function is computationally prohibitive. However, due to the concave characteristic of the energy function [2], it is possible to achieve an energy minimum of the assignment with a gradient descent method. This means that any series of swaps, in which each swap individually reduces the energy of the assignment, will converge in an energy minimum.

Nevertheless, preferably a small number of swaps should be performed to reduce the runtime of the energy minimization. Therefore, the energy minimization is performed iteratively. In each iteration, all pairs of sites (s_i, s_j) with $s_i, s_j \in S$, $i < j$, are examined for swaps that improve the overall energy of the assignment. While examining a pair of sites, we perform the minimal number of swaps that yield an energy minimum for these two sites. This combination of pairwise examination and optimal swapping for each pair results in a short runtime of the algorithm while preserving a linear memory complexity. If no swap is found for all pairs of sites, the iterative minimization process stops, and the resulting assignment is equivalent to the power diagram of S. The necessary number of iterations until the minimization stops is not predictable, but is guaranteed to be finite because of the strictly monotonic decreasing energy function.

The underlying energy function $e(x,s)$ of the minimization is derived from the distance function of the power diagram, for which the additive weight component is omitted:

$$e(x,s) = \|x-s\|^2. \tag{3}$$

This naturally models the compactness and the shape of the bisectors of the regions in the power diagram. To determine whether swapping two points of different sites reduces the energy of the assignment, it is necessary to evaluate their energy before and after the potential swapping. The energy difference Δe for a single point x_i, whose assignment is changed from site s_i to another site s_j, is given by

$$\Delta e(x_i, s_i, s_j) = e(x_i, s_i) - e(x_i, s_j), \tag{4}$$

with $\Delta e > 0$ if the energy of the assignment is reduced. To find the optimal swapping candidates between the points of two different sites, we perform a greedy-like approach with the following steps for each pair of sites (s_i, s_j):

(1) We initialize two max-heap data structures H_i,H_j for storing the points assigned to s_i, and s_j, respectively.

(2) For each point x_i assigned to s_i, we calculate the energy difference $\Delta e(x_i, s_i, s_j)$, and insert x_i into the max-heap H_i with Δe as its heap key.
(3) Likewise, for each point x_j assigned to s_j, we calculate the energy difference $\Delta e(x_j, s_j, s_i)$, and insert x_j into the max-heap H_j with Δe as its heap key.
(4) Now we can access the points of s_i and s_j in descending order, starting with those points that give the largest energy improvement, and evaluate the influence of a swap on the overall energy:
   (a) First, we test if both heaps contain any points. If this is the case, we combine the energy differences of the two maximum elements $max(H_i)$ and $max(H_j)$. If the result is greater than zero, a swap of the assignments of these two elements reduces the total energy, and we proceed with step (b). If the combined energy difference is equal to or less than zero, no swap will reduce the total energy. Hence, we can stop the evaluation of this pair of sites. The same applies if one of the heaps is empty.
   (b) To actually perform the swap, we change the assignment $A : X \rightarrow S$ for the two maximum elements to $A(max(H_i)) := s_j$ and $A(max(H_j)) := s_i$. We then remove these elements from the heap. Finally, we indicate that the assignment is not stable, by setting the corresponding flag to *false*. Then we proceed with step (a).

Algorithm 1: Capacity-constrained power diagram in finite space

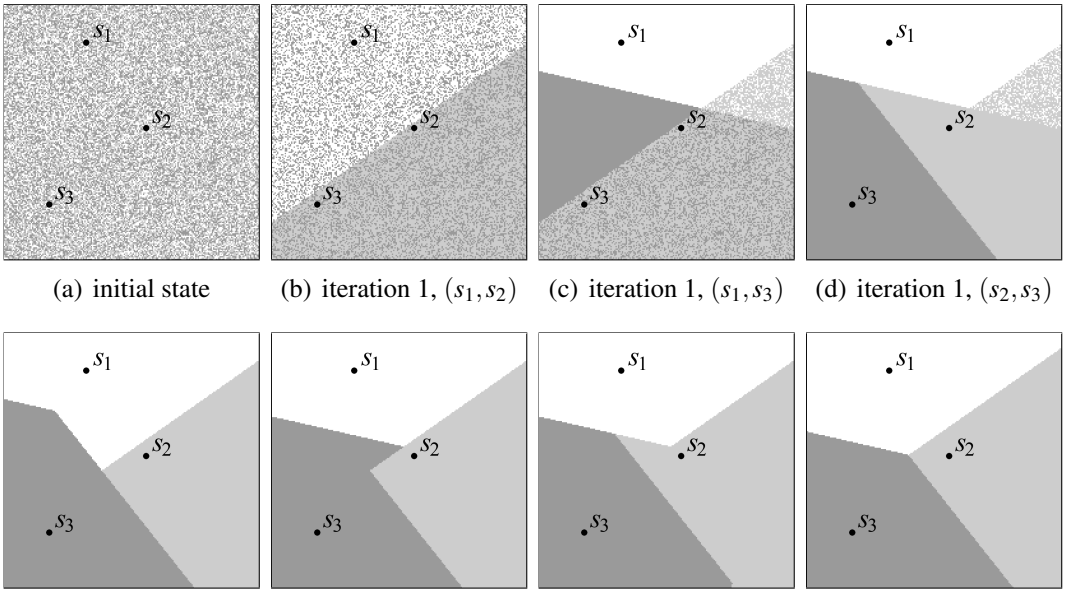
Input: Set S of n sites, Set X of m points, Set $C = \{c(s) | s \in S\}$ of n capacities

Output: Assignment $A : X \rightarrow S$ that represents a power diagram of S for X with $c(s) = |reg(s)|, s \in S, c \in C$

```
Initialize an arbitrary assignment A that fulfills the capacity constraint in C ;
repeat
    stable := true ;
    foreach pair of sites (s_i, s_j) with s_i, s_j ∈ S, i < j do
        Initialize two heap data structures H_i, H_j ;
        foreach point x_i with A(x_i) = s_i do
            insert x_i into H_i with Δe(x_i, s_i, s_j) as its key ;
        foreach point x_j with A(x_j) = s_j do
            insert x_j into H_j with Δe(x_j, s_j, s_i) as its key ;
        while |H_i| > 0 and |H_j| > 0 and max(H_i) + max(H_j) > 0 do
            modify the assignment A to A(max(H_i)) := s_j, and
            A(max(H_j)) := s_i ;
            remove max(H_i) from H_i, and max(H_j) from H_j ;
            stable := false ;
until stable = true ;
```

A further illustration of our algorithm is shown in Figure 2.2. This example consists of three sites with equal capacities, computed on a regular grid of 200 $\times$

200 points in less than on second on consumer hardware. Fig. 2.2(a) presents the random initial state of the assignment. The state after the optimal swapping of the points for the first pair of sites in the first iteration is shown in (b), with the correct orthogonal bisector of the pair of sites (s_1, s_2) in evidence. Similarly, the orthogonal bisectors of the pairs (s_1, s_3) and (s_2, s_3) are formed in (c) and (d), whereby the bisector between (s_1, s_2) is necessarily disrupted due to the swapping between the white and dark grey points of s_1 and s_3 in step (c). In the second iteration (e–g), this pairwise swapping is repeated. Again, the correct orthogonal bisector is formed for each pair, while disrupting other bisectors. However, the interval which is alternately occupied by the three sites is getting smaller from iteration to iteration. Thus, after 77 iterations, a stable state (h), which represents a valid power diagram, is finally achieved.



(a) initial state (b) iteration 1, (s_1, s_2) (c) iteration 1, (s_1, s_3) (d) iteration 1, (s_2, s_3)

(e) iteration 2, (s_1, s_2) (f) iteration 2, (s_1, s_3) (g) iteration 2, (s_2, s_3) (h) result, 77 iterations

FIGURE 1. First steps and final result of the power diagram computation using our swapping algorithm for three sites with equal capacity on a regular grid

2.3. **Properties.** Our algorithm performs swaps of the assignment of points if and only if such a swap decreases the total energy of the assignment. Thus, it is guaranteed that it converges in a stable state where no further swap improves the energy. In this stable state, the assignment is equivalent to a power diagram as proven by Aurenhammer et al. [2].

The simple structure of our algorithm allows for its easy adaption to different finite spaces. The only prerequisite is a function to determine the Euclidean distance between two points, which is used in lines 7 and 9 of the algorithm. Everything else is generic and independent of the underlying space. This also implies that we can easily implement other geometric objects than just points as sites.

The complexity of our algorithm is influenced by the number of sites n and the number of points m. In the worst case, we have to evaluate n^2 pairs of sites in

each iteration. Thereby, we have to initialize a heap data structure and organize $\frac{m}{n}$ points in the heap data structure according to the energy difference of the potential swap, which at the bottom line is equivalent to sorting—note that the worst case for the point heap is $\frac{m}{n}$ due to the fact that the while loop in line 10 stops if just one heap is empty. The resulting complexity is $O(n^2(1+\frac{m}{n}\log\frac{m}{n}))$. By reducing this term we obtain a worst case time complexity for each iteration of $O(n^2+nm\log\frac{m}{n})$. The number of iterations is not directly related to n and m. Roughly speaking, the number of iterations increases/decreases as $\frac{m}{n}$ increases/decreases. Also note, that in practical application, n is by orders of magnitude smaller than m, so the quadratic term is usually dominated by the second linearithmic term. The memory complexity of our algorithm is $O(n+m)$.

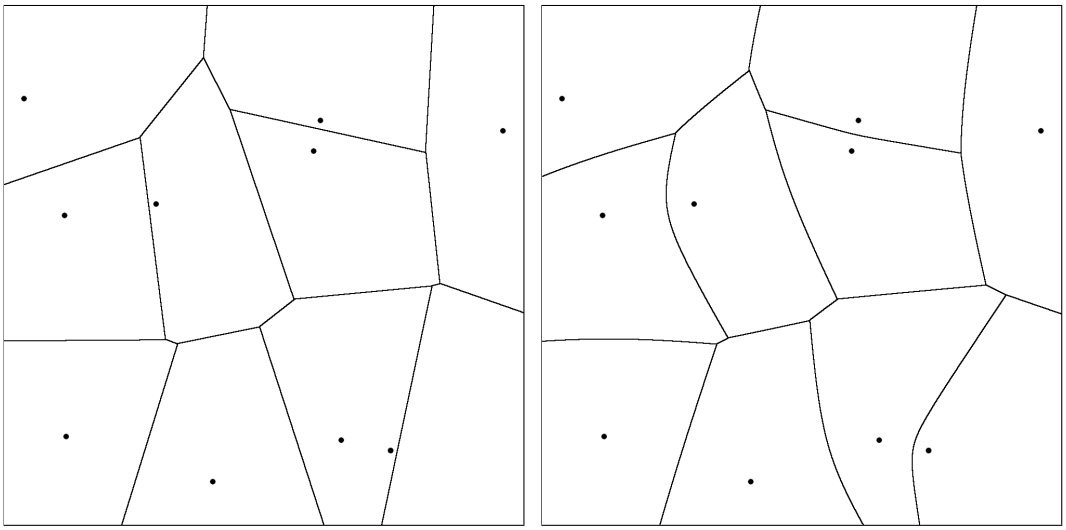
The utilized heap data structure in our algorithm reduces its average runtime due to the fact that only a small fraction of points is swapped, and for this reason we do not have to perform a complete sorting. A further speedup is achieved by introducing a bounding test for reducing the number of pairs of sites and the number of points that are evaluated. Therefore, for each site we construct a hypersphere that encloses all points assigned to that site. From the geometry of the problem it follows that there only exist beneficial swaps in the area where the hyperspheres of two sites overlap. This means that pairs of sites with non-overlapping hyperspheres do not have to be evaluated. Furthermore, points that are not located in the overlapping area do not have to be evaluated. This considerably reduces the number of site pairs and points that have to be tested. The bounding test optimization has been omitted in the foregoing section for a clearer presentation of our approach.

3. Extensions

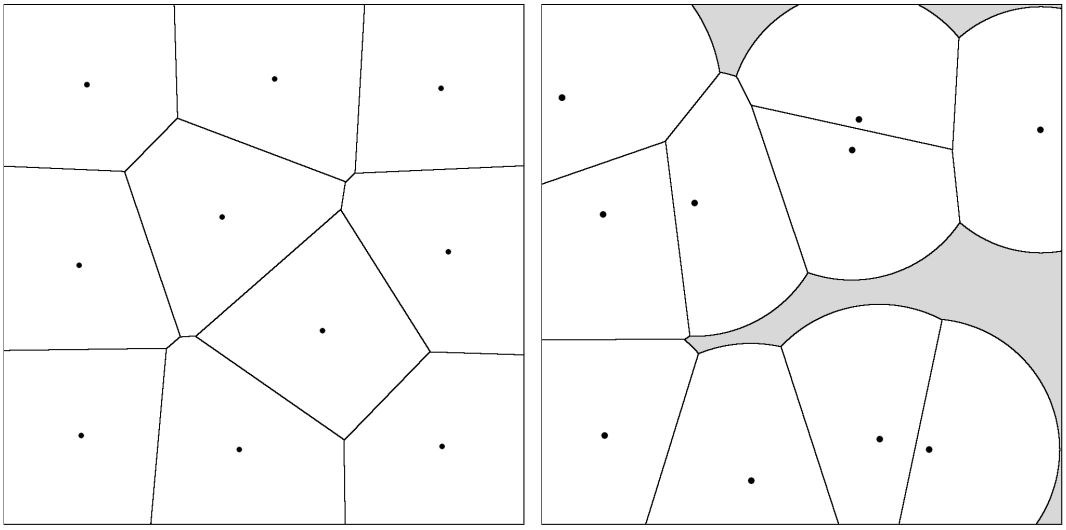
In the previous section we described our approach for computing capacity-constrained power diagrams. Its foundation is the least-squares minimization, which is directly related to the distance function of the power diagram. This section presents extensions of our method which enable the computation of other capacity-constrained Voronoi diagrams beyond the power diagram. These extensions do not affect the computational complexity of our algorithm. Furthermore, all extensions can be combined with each other. Figures 2(b–d) present examples for the extensions in this section, and Figures 2(a) presents a power diagram for comparison. Each of these examples is based on the same initial set of ten sites with equal capacities, computed on a regular grid of 1000×1000 points.

3.1. **Other Energy Functions.**

Our algorithm in Section 2.2 generates a power diagram by minimizing the energy function in Equation 3, which is directly related to the distance function of the power diagram. By replacing this energy function, we can generate other capacity-constrained weighted Voronoi diagrams. The range of Voronoi diagrams that can be generated with our approach is limited to those having an additive weight component in their distance function. Voronoi diagrams with a multiplicative weight component can not be generated. For other energy functions with additive weights it is not guaranteed that there actually exists an Voronoi diagram for any given sets of sites and capacity constraints. But our algorithm will generate such a diagram, if there exists one.



(a) power diagram (b) additively weighted Voronoi diagram



(c) centroidal power diagram (d) power diagram with a void region (grey)

FIGURE 2. Examples for extensions (b-d) of our algorithm in Section 2.2. A power diagram (a) is given for comparison. All diagrams are based on the same initial set of ten sites with equal capacities.

The necessary energy function can be derived directly from the distance function for the intended Voronoi diagram by omitting the weight component. For example, to achieve an additively weighted Voronoi diagram with the L^2 norm we use the following energy function:

$$\delta_{aw} = \|x-s\| - w \quad \Longrightarrow \quad e_{aw}(x,s) = \|x-s\|. \tag{5}$$

In fact, we can use all kinds of energy functions as long as they fulfill the definition of a norm. An example for an additively weighted Voronoi diagram with equal capacities is shown in Figure 2(b). Here the sites are the same as in the power diagram in Figure 2(a).

3.2. **Centroidal Voronoi Diagrams.** A Voronoi diagram in which each site is located in the center of mass of its corresponding region is called a centroidal Voronoi diagram [4]. Such diagrams possess uniformly distributed sites and compact corresponding regions, which are properties that are demanded in many application domains, such as image processing, compression, statistics, logistics, or biology. A centroidal Voronoi diagram can be iteratively computed with Lloyd's method [5] by generating the Voronoi diagram of a set of sites, then moving these sites into the center of mass of the corresponding regions, and repeating these two steps until a stable state is achieved. In fact, this method is also a least-squares minimization, using the same energy function as in Equation 3.

A capacity-constrained centroidal Voronoi diagram can be generated by just applying our algorithm to the generation of the Voronoi diagram in Lloyd's method. However, we achieve a much better convergence if we directly integrate the site relocation into our algorithm. We therefore insert the site relocation after line 13 of the algorithm in Section 2.2, as the last statement inside the foreach loop. This means that immediately after the point assignment has been changed, we relocate both sites. The convergence of this extension is guaranteed because relocating a site always decreases the energy of the assignment. When combining centroidal Voronoi diagrams with energy functions that are not based on the L^2 norm, this convergence guarantee may not apply. However, in our experiments with non-monotonic decreasing energy functions we did not identify a case where our algorithm did not converge. An example for a centroidal power diagram with equal capacities is shown in Figure 2(c). The initial locations of the sites are the same as in the power diagram in Figure 2(a).

3.3. **Void Region.** The Voronoi diagram of a set X of points for a set S of sites is defined as a partition into disjoint subsets, which means that every point in X is assigned to exactly one site in S. This implies that the sum of the capacities in C is equal to the number of points in X. If we want to create a Voronoi diagram where the sum of the capacities is smaller than the number of points, we have to add a special site that gathers all points that are not assigned to a site in S. We call the points that are assigned to this special site the *void region*.

The void region is seamlessly integrated into our algorithm by inserting it as an additional site into our set of sites. The site location of the void region is undetermined, and the energy of a point in X that is assigned to the void region is always zero, $e(x, s_{void}) = 0$. As result, the void regions gathers all leftover points with high energy values. An example for a power diagram with a void region is shown in Figure 2(d). Here, the total capacity equals 90% of the number of points, and again all sites have the same capacity. The void region is colored grey, and covers the remaining 10% of the area.

3.4. **Density Functions.** In Section 2 we assume that each point $x \in X$ has a capacity of one. Hence, the capacity of a site is equal to the number of points assigned to this site. This concept is extended by adding a density function $\delta(x)$ that determines the capacity of each point individually. Now the *actual capacity* $c'(s)$ of a site $s \in S$ is determined by the sum of the capacities of the points assigned to this site, $c'(s) = \sum \delta(x)$ with $A(x) = s$. Admittedly, this actual capacity $c'(s)$ may greatly differ from the given capacity $c(s)$ if we restrict ourselves to assignment

swaps. The reason therefore is that now each swap can change the actual capacity of a site if the two points do not have equal density values. To reduce this inevitable capacity difference, we have to introduce an additional step into the algorithm that performs a capacity balancing between the sites. Here, we change the assignments of individual points that have minimal impact on the total energy. This capacity balancing competes with the energy minimization, which undermines the convergence guarantee of our algorithm and may violate the capacity constraint. A detailed description of our current approach for this problem is beyond the scope of this paper and will be addressed in future work.

4. Conclusion

We presented an approach for the computation of capacity-constrained Voronoi diagrams in finite spaces. In contrast to other approaches, which iteratively adjust a weighted Voronoi diagram until the capacity constraint is fulfilled, we start with an arbitrary partition that already fulfills the capacity constraint, and iteratively optimize this partition until it represents a weighted Voronoi diagram. The foundation for this optimization is the swap of assignments of points to sites based on an energy function that is directly related to the distance function of the Voronoi diagram. By the restriction to assignment swaps, it is guaranteed that the capacity constraint is preserved in all steps of the optimization, including the result.

The straightforward structure of our algorithm makes it easy to implement, flexible, and fast. For example, the generation of the Voronoi diagrams in Figure 3, showing different extensions of our algorithm, have each been computed with a few hundred or thousand iterations in less than one minute on consumer hardware.

References

1. Franz Aurenhammer, *Voronoi diagrams—a survey of a fundamental geometric data structure*, ACM Computing Surveys **23** (1991), no. 3, 345–405.
2. Franz Aurenhammer, Friedrich Hoffmann, and Boris Aronov, *Minkowski-type theorems and least-squares clustering*, Algorithmica **20** (1998), no. 1, 61–76.
3. Michael Balzer and Oliver Deussen, *Voronoi treemaps*, Proceedings of the IEEE Symposium on Information Visualization (Minneapolis, MN, USA) (John T. Stasko and Matthew Ward, eds.), IEEE Computer Society, October 2005, pp. 49–56.
4. Qiang Du, Vance Faber, and Max Gunzburger, *Centroidal Voronoi tessellations: Applications and algorithms*, SIAM Review **41** (1999), no. 4, 637–676.
5. Stuart P. Lloyd, *Least square quantization in PCM*, IEEE Transactions on Information Theory **28** (1982), no. 2, 129–137.
6. Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu, *Spatial tessellations: Concepts and applications of Voronoi diagrams*, 2nd ed., John Wiley and Sons Ltd., May 2000.
7. René Reitsma, Stanislav Trubin, and Eric N. Mortensen, *Weight-proportional space partitioning using adaptive Voronoi diagrams*, Geoinformatica **11** (2007), no. 3, 383–405.

Department of Computer and Information Science, University of Konstanz, Germany

E-mail address: {michael.balzer,daniel.heck}@uni-konstanz.de