Chapter 4

Point Labeling: Label-Size Maximization

When placing labels on maps, maximizing the (weighted) number of features that receive a label is certainly the aim that plays the greatest role in practice. However, one might think of other, for instance technical applications where holes must be drilled next to a given number of points on a piece of metal, and the size of these holes is to be maximized. This is an example of the label-size maximization problem that we consider in this chapter.

When comparing the complexity status of label-number and label-size maximization, it is difficult to decide which problem is harder. Label-size maximization can be solved in polynomial time in some special cases: efficient algorithms are known for two label candidates per feature [FW91] and, if the label candidates overlap in a certain way, for any constant number of label candidates per feature [PZC98, SvK99], and even for an infinite number of label candidates per feature [KSY99]. On the other hand the problem of maximizing the number of points with axis-parallel rectangular labels can be approximated arbitrarily well (see Section 3.3.3) while maximizing the size of square labels cannot be approximated better than by a factor of 1/2 [FW91].

The label-size maximization problems that have been considered so far are the following: labeling points with circles, squares or other regular polygons [FW91, DMM$^+$97] and labeling axis-parallel line segments with axis-parallel rectangles that have the same length as the segment they label and must touch or contain the segment [PZC98, SvK99, KSY99]. In this chapter we will consider labeling points with axis-parallel rectangles and with circles.

4.1 Rectangular Labels

Formann and Wagner proposed an approximation algorithm that maximizes the size of uniform axis-parallel square labels. It is optimal in respect to both, its approximation factor of 1/2 and its running time of $O(n \log n)$ [FW91, Wag94].

Here n refers to the number of points, each of which has four label candidates. For the same problem, there is an algorithm that keeps the theoretical optimality of the approximation algorithm, but performs close to optimal in practice [WW97].

It is obvious that the approximation result of Formann and Wagner also holds for uniform rectangular labels; the coordinate system can always be scaled such that these labels become squares. For rectangles of arbitrary width, however, or for rectangles of arbitrary height and width, no approximation algorithms are known, not even heuristics have been suggested. However, the label-size maximization problem can be reduced to the decision problem if the number of *conflict sizes* can be bounded, i.e. the scaling factors for which label candidates start to intersect. Then one can do a binary search on a sorted list of these conflict sizes. For each step of the search, one would compute the current candidate conflict graph and call an algorithm for the decision problem for the current scaling factor. Any algorithm for the label-number maximization problem could be employed; if it does not find a complete labeling, we return false, and the binary search continues with a lower value, with a higher value otherwise.

In practice, however, the number of available font sizes is usually a small constant, hence a binary search on a list of conflict sizes is not necessary. One can simply start with the smallest font, call label-number maximization, and repeatedly increase the font size as long as the number of unlabeled points is tolerably small.

In [DMM$^+$97] Doddi et al. suggest such a bi-criteria algorithm that mediates between the two fundamental optimization problems, namely label-size and label-number maximization. Given an $\varepsilon > 0$, the algorithm labels at least a $(1-\varepsilon)$ - fraction of the points with axis-parallel uniform square labels of size at least $n_{\text{opt}}/(1+\varepsilon)$, where n_{opt} is the edge length of the squares in an optimal solution of all points. The algorithm puts $1/\varepsilon$ equidistant markers on each label edge and places the label such that one of the markers coincides with the point to be labeled.

In the same paper, Doddi et al. give approximation algorithms that maximize the size of square labels of arbitrary orientation and of circular labels, again under the restriction that all labels are uniform, i.e. of equal size. The approximation factors of their algorithms are approximately $\frac{1}{37}$ for squares and $\frac{1}{30}$ for circles. In the next section we will improve the approximation factor of their algorithm for circular labels by about 50%.

4.2 Circular Labels

This section is joint work with Tycho Strijk, Universiteit Utrecht.

When labeling points, labels are usually restricted to axis-parallel rectangles which (a) have to touch the point they label, and (b) must not intersect any other label. Condition (a) has often been further restricted in that one of a label's corners must coincide with the point to be labeled. In this section we restrict ourselves to a different label shape, namely circles of uniform size, while keeping conditions (a) and (b). We *label* a point by attaching a circle to it such that the circle's boundary contains the point. Our objective is to find the largest real d_{opt}, which still allows us to label *all* given points with non-overlapping circles of diameter d_{opt}. We consider our labels to be open circles, thus they may touch other points or labels.

In considering a set of three points in general position, it is clear that approximating the maximum size of circular labels cannot be reduced to the same problem for square labels. While the solution in the former case is bounded (linearly in the diameter of the point set), it is unbounded in the latter.

We show that even deciding whether a set of points can be labeled with unit circles is NP-hard, see Section 4.2.5. This settles an open question raised in [DMM$^+$97]. The same proof implies that there is a constant $\delta < 1$ such that it is NP-hard to label points with circles of diameter greater than $\delta \cdot d_{\text{opt}}$. Nevertheless, the maximization problem has already been approximated. Doddi et al. suggested a simple algorithm that labels points with circles whose diameter is at least $1/(4(2+\sqrt{3})) \approx 1/14.93$ times the optimum and takes $O(n \log n)$ time [DMM$^+$97]. However, a careful revision of their proof, see Section 4.2.1, shows that the approximation factor of their algorithm is actually worse by a factor of 2; i.e. the label diameter is guaranteed to be at least $1/(8(2+\sqrt{3})) \approx 1/29.86$ times the optimum. In this paper, we present an algorithm with an approximation factor of 1/19.59. While the analysis that yields this factor becomes more involved, the algorithm remains simple.

Both algorithms first determine the smallest diameter D_3 of any three-point subset of the input points. This can be done in $O(n \log n)$ time [DLSS95]. D_3 is needed to compute the diameter of the labels, which in both cases is a constant fraction of D_3. The observation that no point set of more than two points can be labeled with circles of diameter greater than $2(2+\sqrt{3})D_3$ yields the respective approximation factors.

Like the algorithm of Doddi et al., when labeling a point our algorithm only needs to know the location of the point's closest neighbor in the set of input points. However, while Doddi et al. maximize the distance between the labels of a pair of closest neighbors, we minimize it in order to use space more efficiently. This implies that they only need to know the *direction* of the closest neighbor while we also need its *distance.* Another difference is that while they label the points in arbitrary order, we exploit this order and process the points in pairs of increasing distance. In order to build and access the data structure that supplies us with this order we need no more than $O(n \log n)$ time in total. Thus our algorithm runs in $O(n \log n)$ time. It requires linear storage.

This section is structured as follows. In Section 4.2.1, we sketch the algorithm of Doddi et al.. In Section 4.2.2 we formalize our main ideas. Then, in Section 4.2.3 we present our algorithm, analyze it in Section 4.2.4, and finally present our NP-hardness proof in Section 4.2.5.

4.2.1 Previous Work

For a (finite) set S of points in the plane, Doddi et al. define the *diameter* diam(S) the usual way as the maximum distance of any two points in S. They define the *k-diameter* $D_k(S)$ to be the minimum diameter over all k-element subsets of S. Then they make the following two observations. In our description we will abbreviate $D_3(S)$ by D_3 where appropriate.

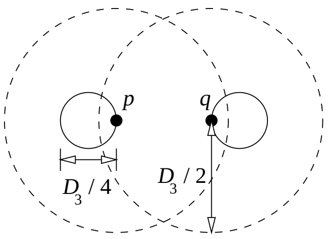

Figure 4.1: Label placement according to the algorithm of Doddi et al..

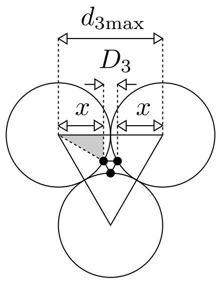

Figure 4.2: Optimal label placement for three points.

1. The open circle centered at a point $p \in S$ with radius $D_3/2$ contains at most one other point $q \in S - \{p\}$. Due to symmetry, an open circle of the same diameter centered at q only contains p and q. This allows p and q to be labeled with labels of diameter $d' = D_3/4$ as in Figure 4.1. Given the distance of p and q to other points in S, it is obvious that labels of other points cannot overlap those of p and q.

2. The maximum label diameter $d_{\text{opt}}(S)$ of any set S of more than two points cannot exceed the maximum label diameter $d_{3\max}$ of three points at pairwise distance $D_3(S)$, see Figure 4.2. Doddi et al. compute $d_{3\max}$ to be $(2+\sqrt{3})D_3$, but this is incorrect: we have $d_{3\max} = 2x + D_3$, where $x = (d_{3\max}/2) \cdot \cos(\pi/6)$, see the shaded triangle with a right angle and a 30° angle in Figure 4.2. Thus we obtain $d_{3\max} = 2(2+\sqrt{3})D_3 \approx 7.46\ D_3$ as an upper bound for d_{opt}.

Combining these observations yields the approximation factor $d'/d_{\text{opt}} \geq 1/(8(2+\sqrt{3})) \approx 1/29.86$ of the algorithm of Doddi et al..

4.2.2 Preliminaries

We formalize the idea of the free space around a point as follows.

Definition 4.1 *Let $C_{m,r}$ be an open circle with radius r centered at m. We say that two points are a* pair of closest neighbors *if each is a closest neighbor of the other. Given two points $p, q \in S$, we denote the* point-free zone *of p and q by $Z_{\text{free}}(p,q) = (C_{p,D_3} \cap C_{q,D_3}) \cup C_{p,d} \cup C_{q,d} \subseteq \mathbb{R}^2$ where $d = d(p,q)$ is the Euclidean distance of p and q.*

The definition is illustrated in Figure 4.3. We show that the point-free zone of a pair of closest neighbors $\{p, q\}$ does not contain any other point of S. This will enable us to use part of the zone for labeling p and q.

Lemma 4.2 *$Z_{\text{free}}(p, q) \cap S = \{p, q\}$ for any pair $\{p, q\}$ of closest neighbors in S.*

Proof. Suppose $Z_{\text{free}}(p, q)$ contains a point $t \in S \backslash \{p, q\}$. Then $t \in C_{p,D_3} \cap C_{q,D_3}$ or $t \in C_{p,d} \cup C_{q,d}$. In the first case the diameter of $\{p, q, t\}$ would be less than D_3; a contradiction to the definition of D_3. The second case would contradict $\{p, q\}$ being closest neighbors. ◶

Note that Doddi et al. implicitly also used the concept of a point-free zone, namely the union of the dashed circles $C_{p,D_3/2}$ and $C_{q,D_3/2}$ depicted in Figure 4.1. However, their zone is always contained in our point-free zone Z_{free}, independently of d. This helps us to place larger labels. Let d_{algo} be the diameter of the labels our algorithm is going to place.

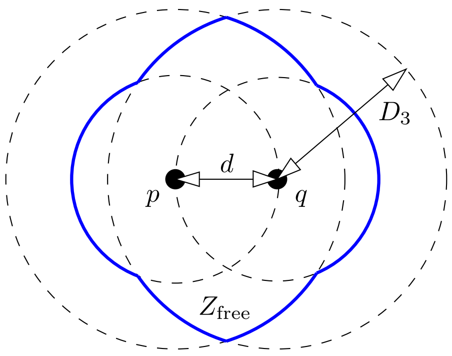

Figure 4.3: The point-free zone Z_{free} of p and q.

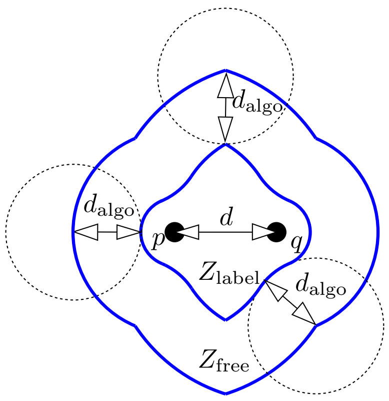

Figure 4.4: The label zone Z_{label} of p and q lies inside Z_{free}.

Definition 4.3 *Given two points $p, q \in S$ and a real number $d_{\text{algo}} < D_3$. Then we denote the* label zone *of p and q by*

$$Z_{\text{label}}(p, q; d_{\text{algo}}) = Z_{\text{free}}(p, q) \ominus C_{0,d_{\text{algo}}} = Z_{\text{free}}(p, q) - \bigcup_{x \in \mathbb{R}^2 - Z_{\text{free}}(p,q)} C_{x,d_{\text{algo}}}$$

where $\ominus$ is the Minkowski subtraction operator and 0 is the origin.

In other words, the label zone is the erosion of the point-free zone with a disk of radius d_{algo}. The definition is illustrated in Figure 4.4.

Lemma 4.4 *If we label a pair of closest neighbors $\{p, q\}$ with circles of diameter d_{algo} that are contained in the label zone $Z_{\text{label}}(p, q; d_{\text{algo}})$, then these labels cannot overlap the label of any other point $t \in S - \{p, q\}$.*

Proof. Suppose the label of t overlaps that of p. Lemma 4.2 tells us that $t \notin Z_{\text{free}}(p,q)$. Then the definition of the label zone ensures that $C_{t,d_{\text{algo}}}$ does not intersect $Z_{\text{label}}(p,q;d_{\text{algo}})$. Observing that t's label is contained in $C_{t,d_{\text{algo}}}$ and p's label in $Z_{\text{label}}(p,q;d_{\text{algo}})$ contradicts our assumption. ❍

The question is, of course, how large we can choose d_{algo} so that the labels of any pair of closest neighbors fit into their label zone—independently of their distance. This question is dealt with in Section 4.2.4. Let us suppose for the moment that d_{algo} can be expressed as a fraction of D_3. The next section answers two other important questions, namely how to place the labels, and in which order.

4.2.3 Algorithm

Our algorithm proceeds as described in Figure 4.5. The value of $D_3(S)$ is computed with the algorithm of Datta et al.[DLSS95]. In contrast to Doddi et al. who place the labels of two neighboring points as far apart from each other as possible, we label p and q such that their labels are as close as possible. This means that they will touch each other as in Figure 4.6 if $d(p,q) \le 2d_{\text{algo}}$. The vectors $\vec{p}$ and $\vec{q}$ denote the coordinates of p and q in the plane. We place the center $\vec{m_q}$ of q's label at $\vec{m_q} = \vec{p}/4 + 3\vec{q}/4 + \vec{a}$. The vector $\vec{a}$ is perpendicular to $\overline{pq}$ and oriented so that it points to the left of $(\vec{p} - \vec{q})$. The length of $\vec{a}$ is $\|\vec{a}\| = \sqrt{d^2_{\text{algo}}/4 - d^2(p,q)/16}$. Correspondingly, the center of p's label is placed at $\vec{m_p} = \vec{q}/4 + 3\vec{p}/4 - \vec{a}$. We call the union of these two (open) labels the *label space* of p and q. If there are unlabeled points left after executing the while-loop, we label them arbitrarily.

```
LABEL_POINTS_WITH_CIRCLES($S$)

compute $D_3(S)$
$d_{\text{algo}} := 0.381\ D_3(S)$
**while** $|S| > 1$
      choose $\{p,q\} \subseteq S$ with $d(p,q)$ minimal
      **if** $d(p,q) \ge 2d_{\text{algo}}$ **then** exit while-loop
      label $p$ and $q$ as in Figure 4.6
      $S := S \setminus \{p,q\}$
**end**
**for all** $x \in S$ **do** label $x$ arbitrarily **end**
**return** all label positions
```

Figure 4.5: Our algorithm.

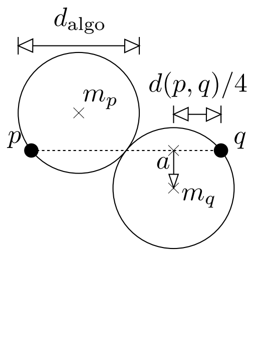

Figure 4.6: Labeling a pair of points.

Lemma 4.5 *Given a set S of n points and a label diameter d_{algo} such that the label space is contained by the label zone for any pair of points in S, then the labels our algorithm places do not intersect.*

Proof. The fact that no two labels overlap follows from the order in which we process the points. It is clear that the first pair $\{p, q\}$ is a pair of closest neighbors. Due to Lemma 4.4, we know that we do not constrain the labeling of any other point in S when we label such a pair within its label zone. In other words, if we remove $\{p, q\}$ from S, then we can ignore p and q as well as their labels for solving the remaining problem. The next pair of points will be a pair of closest neighbors in the reduced set S. Thus Lemma 4.4 applies to this pair as well.

After we leave the while-loop, there may be unlabeled points left. For each such point x there are two possibilities. Either its closest neighbor in the original set is at least $2d_{\text{algo}}$ away. Or all points y with $d(x, y) < 2d_{\text{algo}}$ have been labeled before, since each had a closer neighbor z. In either case the labeling of x is not constrained by any previous label placement. Hence we can label x arbitrarily. ◶

Lemma 4.6 *The algorithm can be implemented such that it labels a set S of n points in $O(n \log n)$ time with linear space.*

Proof. Our algorithm labels S in three phases. In the first phase, we compute D_3 in $O(n \log n)$ time [DLSS95]. We need D_3 to compute the diameter $d_{\text{algo}} = 0.381\ D_3$ of the labels we are going to place, see Section 4.2.4.

In the second phase, we set up a simple data structure that will answer a limited closest pair query; limited in the sense that we only need to know pairs of points closer than $2d_{\text{algo}}$ in the Euclidean metric. We call these *pairs of relevant neighbors.* An axis-parallel rectangle of size $2d_{\text{algo}} \times d_{\text{algo}}$ contains at most two input points since it fits into a circle of diameter D_3. Thus an axis-parallel square of edge length $4d_{\text{algo}}$ centered at a point $p \in S$ contains at most twelve input points apart from p. The relevant neighbors of p are obviously among these. This observation enables us to collect all pairs of relevant neighbors with a sweep-line—or rather sweep-window—approach.

As usual we use two data structures: an event-point queue as horizontal structure and a sweep-line status as vertical structure. Our sweep window is a vertical strip of width $2d_{\text{algo}}$ and moves over the plane from left to right. Its right border line r stops at each event point. We have two kinds of events: when an input point $p = (x_p, y_p)$ enters the window (r is at x_p) and when p leaves the window (r is at $x_p + 2d_{\text{algo}}$). When p enters the window we want to report efficiently all points in the window whose y-coordinate is less than $2d_{\text{algo}}$ from y_p. For this purpose the sweep-window status is implemented by a balanced binary tree on the y-coordinates of the points in the window. For later on, we insert each pair $\{p, q\}$ that is reported during the sweep into a priority queue according to its Euclidean distance $d(p, q)$ if $d(p, q) < 2d_{\text{algo}}$. Our sweep takes $O(n \log n)$ time and uses linear space.

In the third phase, we repeatedly extract the minimum $\{p, q\}$ of the priority queue, label p and q with circles of diameter d_{algo} as in Figure 4.6, and delete all pairs containing either p or q from the queue. The remaining points are labeled arbitrarily in constant time per point. Phase 3 can also be done in $O(n \log n)$ steps.

The running time of the three phases sums up to a total of $O(n \log n)$. The necessary data structures require linear space. ❍

4.2.4 Analysis

Given a pair $\{p, q\}$ of closest neighbors in S, our objective now is to compute the maximum radius r of their labels so that the label space is still contained in the label zone $Z_{\text{label}}(p, q; 2r)$ of p and q. Since this radius will only depend on the distance d of p and q, we want to find the minimum $r_{\min}$ of $r(d)$, set d_{algo} to a value slightly less than $2r_{\min}$ and run our algorithm. Lemma 4.5 guarantees that no two labels will intersect then.

Since we place the labels of p and q symmetrically, it is enough to analyze the placement of q's label. We consider two cases depending on the distance of p and q.

In case $d \leq D_3/2$, the point-free zone $Z_{\text{free}}(p, q)$ is the intersection of C_{p,D_3} and C_{q,D_3}. The corresponding label zone $Z_{\text{label}}(p, q; 2r)$ is the intersection of C_{p,D_3-2r} and C_{q,D_3-2r}. As described in the previous section, the label has its center point m_q on a line h, normal to the line connecting p and q. This normal line has distance $d/4$ to q. The distance of m_q to the line $\overline{pq}$ is $\sqrt{r^2 - d^2/16}$ using Pythagoras' rule.

The radius of q's label is maximized when the label touches the boundary of the label zone, i.e. the circle C_{p,D_3-2r}. We use the property that the touching point of two circles always lies on the line through their centers, see Figure 4.7.

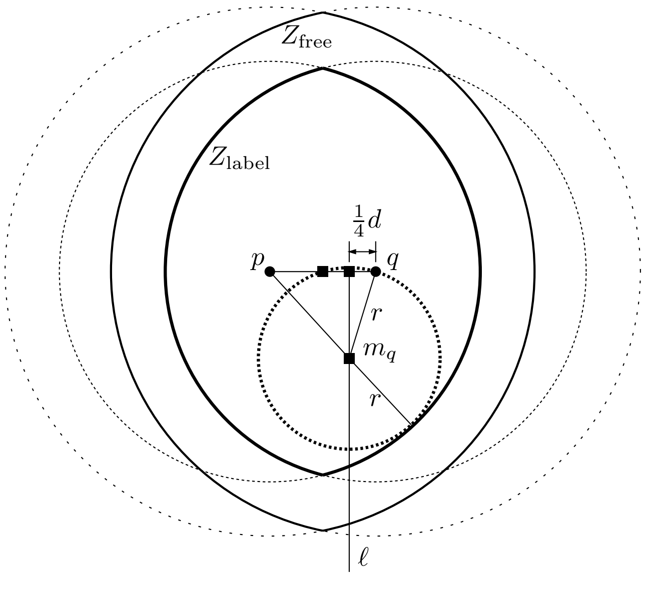

Figure 4.7: Point-free zone Z_{free} and label zone Z_{label} of p and q for $d \leq D_3/2$.

This observation yields the equality

$$d(p, m_q) + r = D_3 - 2r. \tag{4.1}$$

We use that $d(p, m_q) = \sqrt{(\frac{3}{4}d)^2 + (r^2 - d^2/16)} = \sqrt{d^2/2 + r^2}$. The resulting equation

$$\sqrt{d^2/2 + r^2} = D_3 - 3r$$

is quadratic and has two solutions. The solution valid for our problem is

$$r = \frac{3D_3 - \sqrt{D_3^2 + 4d^2}}{8}. \tag{4.2}$$

We will use the notation $\hat{d} = d/D_3$ and $\hat{r} = r/D_3$ to obtain less complicated formulas and to express the fact that the formulas can also be obtained as follows: first scale the point set by a factor $1/D_3$, then determine the optimum label size, and after that scale by a factor D_3 to the original size. As a result Equation (4.2) is simplified to

$$\hat{r} = \frac{3 - \sqrt{1 + 4\hat{d}^2}}{8}. \tag{4.3}$$

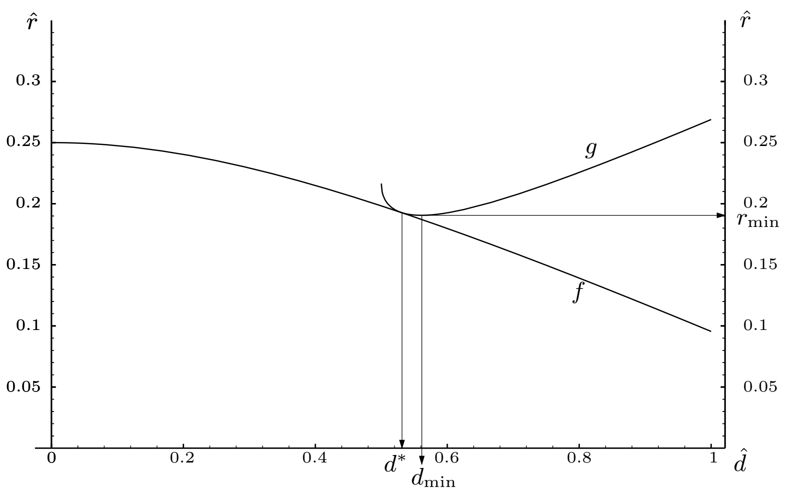

Figure 4.8: The graph of $r(d)$ is determined by the functions f and g. For $d < d^* \approx 0.53\ D_3$, the value of r is determined by f, otherwise by g. The minimum $r_{\min} \approx 0.19\ D_3$ of $r(d)$ is reached at $d_{\min} \approx 0.56\ D_3$.

Graph f in Figure 4.8 depicts Equation (4.3) as a function of $\hat{d}$.

The case $d > D_3/2$ is more difficult. In this case, the point-free zone $Z_{\text{free}}(p, q)$ is the union of three areas, namely $C_{p,D_3} \cap C_{q,D_3}$, $C_{p,d}$, and $C_{q,d}$, see Figure 4.9.

Accordingly, the boundary of $Z_{\text{label}}(p, q; 2r)$ consists of arc segments that are part of the circles C_{p,D_3-2r}, C_{q,D_3-2r}, $C_{p,d-2r}$, $C_{q,d-2r}$ and four circles with

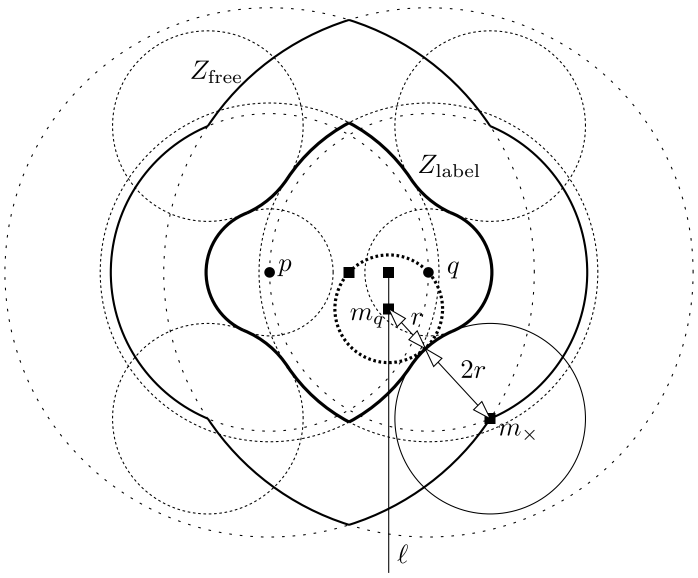

Figure 4.9: The point-free zone Z_{free} and the label zone Z_{label} of p and q for the case $d > D_3/2$. The bold dotted label touches the circle centered at $m_\times$.

radius $2r$ centered at the intersections of C_{p,D_3} with $C_{q,d}$ and at the intersections of C_{q,D_3} with $C_{p,d}$. One of these last four circles is $C_{m_\times,2r}$ whose center $m_\times$ lies at an intersection of C_{p,D_3} and $C_{q,d}$, see Figure 4.9.

It turns out that a label with maximum radius inside the label zone always touches either C_{p,D_3-2r} or $C_{m_\times,2r}$. This depends on the value of d. There is a real $d^* = \frac{\sqrt{1+\sqrt{33}}}{2\sqrt{6}} \cdot D_3 \approx 0.53\, D_3$ such that for $d \leq d^*$ the label touches C_{p,D_3-2r} and for $d \geq d^*$ it touches $C_{m_\times,2r}$.

The values of r for which the label touches C_{q,D_3-2r} have already been computed, see Equation (4.3). The values, for which the label touches $C_{m_\times,2r}$, are computed similarly as follows. The line connecting the center points m_q and $m_\times$ intersects the touching point of the two circles. This gives rise to the equation

$$d(m_q, m_\times) = 3r, \tag{4.4}$$

see Figure 4.9. If we put the origin of our coordinate system at q and let the negative x-axis contain p, then we get

$$m_\times = \left(\frac{1 - 2\,\hat{d}^2}{2\,\hat{d}} D_3,\ -\frac{\sqrt{4 - \hat{d}^{-2}}}{2} D_3 \right),\ \ m_q = \left(-\frac{1}{4}\hat{d} \cdot D_3,\ -\sqrt{\hat{r}^2 - \frac{\hat{d}^2}{16}} \cdot D_3 \right).$$

The equation $d(m_q, m_\times) = 3r$ has the following solution for our problem:

$$\hat{r} = \frac{\sqrt{8 - \hat{d}^{-2} + 8\,\hat{d}^2 - \frac{\sqrt{1 - 16\,\hat{d}^2 + 48\,\hat{d}^4}}{\hat{d}^2}}}{8\sqrt{2}}. \tag{4.5}$$

Graph g in Figure 4.8 depicts Equation (4.5) as a function of $\hat{d}$. The graph of $r(d)$ equals $f(d)$ for $d < d^*$ and $g(d)$ otherwise. The minimum $r_{\min}$ of $r(d)$ is reached at the minimum of g since f decreases monotonically for $d \leq d^*$ and g has a negative derivative at $d = d^*$. A numeric computation shows that g has its minimum value when $d \approx 0.56085\ D_3$. The corresponding minimum value of g and thus of $r(d)$ is $r_{\min} \approx 0.190526\ D_3$.

Theorem 4.7 *Our algorithm labels a finite point set S with circles of diameter $d_{\text{algo}} = 0.381\ D_3(S)$. The approximation factor γ is $0.381/(2(2+\sqrt{3})) \approx 1/19.59$.*

Proof. When we label the points with circles of diameter $2r_{\min}$, we know that the label space of any pair of closest neighbors will be contained in its label zone. Then Lemma 4.5 ensures that for a label diameter $d_{\text{algo}} = 0.381\ D_3 < 2r_{\min}$, our algorithm will label all points with non-overlapping labels. The approximation factor is the ratio of the upper bound $2(2+\sqrt{3})D_3$ (see Figure 4.2) and the diameter d_{algo} of the labels we place. ◶

It is clear that any set of congruent equilateral triangles will force our algorithm to produce a labeling with circles of diameter $\gamma \cdot d_{\text{opt}}$ if the triangles are spaced appropriately. However, there are also examples with a smaller optimum labeling where the algorithm performs better. The triangular lattice formed by the centers of a densest disk packing, for example, has an optimal labeling with circles of diameter $d_{\text{opt}} = D_3$. Here our algorithm yields a ratio of $d_{\text{algo}}/d_{\text{opt}} = 0.381$.

4.2.5 NP-Hardness

In this section we show that deciding whether a set of points can be labeled with unit circles is NP-hard. This answers an open question raised by Doddi et al. [DMM$^+$97]. Our proof also implies that there is a constant $\delta < 1$ such that it is NP-hard to label points with circles of diameter greater than $\delta \cdot d_{\text{opt}}$. Consequently no polynomial-time approximation scheme exists. The proof is by reduction from *planar* 3-SAT. For a Boolean formula of planar 3-SAT type the variable-clause graph is planar. In this graph, the nodes are the variables and clauses of the formula, and there is an edge between a variable node and a clause node if the variable occurs in the clause. The planarity of the variable-clause graph helps to simplify the proof. The same idea is used in Knuth and Raghunathan's proof of the NP-hardness of the Metafont-labeling problem [KR92].

In the Metafont-labeling problem and other label-placement problems studied previously,[FPT81, FW91, MS91] every label can only be placed in a constant number of positions. In our case, there are infinitely many ways to label a point with a circle. This relaxation could potentially make circle labeling polynomially solvable (even given $\mathcal{P} \neq \mathcal{NP}$) and thus simpler than the discrete label-placement problems studied before, just as real-valued linear programming is simpler than zero-one linear programming. Of course the previous

NP-hardness proofs can be modified to allow a certain continuum of feasible label position in the vicinity of the original discrete positions. In [MS91] this was achieved by adding dummy points that do not receive any label; [IL97] uses a similar strategy.

In our reduction, we do not use dummy points, but restrict the infinite number of potential label positions of all points to at most three by using *immobilizing clusters*. These are special gadgets that consist of three points that must be labeled in a *unique* way. This approach is also different from the other two NP-hardness proofs for label-placement problems with an infinite number of label positions per point [IL97, vKSW99]. We take advantage of the special geometry of circles.

Another major difference to all other label-placement problems we know of is the fact that it is not clear whether circle labeling is in $\mathcal{NP}$. We do not know whether there is always a polynomial encoding of a solution, even if the input points have rational coordinates.

Theorem 4.8 *It is NP-hard to decide whether a set of points can be labeled with unit circles.*

Proof. We encode the variables and clauses of a Boolean formula ϕ of planar 3-SAT type by a set of points such that all points can be labeled if and only if ϕ is satisfiable, i.e. if there is a variable assignment such that all clauses evaluate to true. Since Lichtenstein showed that planar 3-SAT is NP-hard [Lic82], it follows that circle labeling is NP-hard as well. Note that the variables and clauses of a planar 3-SAT formula can be embedded in the plane as in Figure 4.10 where all variables lie on a horizontal line and all clauses are represented by *non-intersecting* three-legged combs [Lic82].

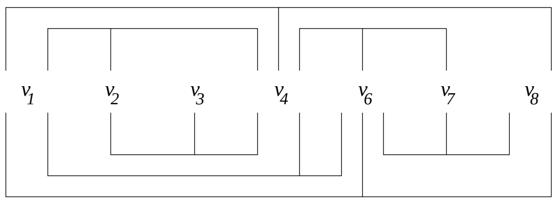

Figure 4.10: Embedding of a planar 3-SAT formula.

The main observation leading to our proof is the following. Given three equidistant points on a line, there are exactly two ways to label these points optimally, see Figure 4.11. Since all labels have diameter 1 here, some basic geometry shows that this distance must be $(1 - \sqrt{2\sqrt{3} - 3})/2 \approx 0.159$. This observation gives us a means to encode the Boolean values of a variable in the planar 3-SAT formula ϕ that we want to reduce to a set of points.

The gadgets of our reduction are the clusters for variables and three-legged combs for clauses. In order to be able to connect a variable v to all clauses in which it occurs, we model v not by one but by several *variable clusters* on a horizontal line h as in Figure 3.58. Note that the cluster-to-cluster distance of $1 + \sqrt{2\sqrt{3} - 3}$ (from midpoint to midpoint) is chosen such that every second cluster must be labeled the same way. The value of v is represented by the label positions of the leftmost cluster on h (according to Figure 4.11). We call this cluster and every second cluster to its right *odd*. Accordingly, all other clusters are *even*. Then the label positions of all odd clusters encode the value of v and all even clusters that of $\neg v$. This differentiation is important for connecting v to the clauses in which it occurs. Each connection depends on whether v is negated in that clause or not.

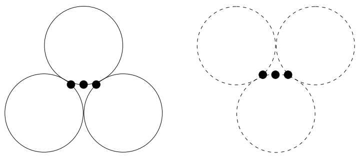
Figure 4.11: A variable cluster and the label placements encoding *true* and *false*.

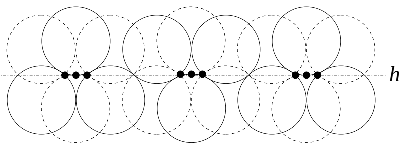

Figure 4.12: Rows of variable clusters model a variable; the label positions of the leftmost cluster determine the variable's Boolean value.

The central idea for modeling the clauses is that we restrict the possible label positions of all points (except one) to a maximum of two. To achieve this, we use *immobilizing clusters* that can only be labeled in one specific way. They consist of three points at pairwise distance $1/(4 + 2\sqrt{3})$, see Figure 4.2. In order to distinguish these auxiliary points from the others, we use the term *active points* for all clause points with at least two possible label positions.

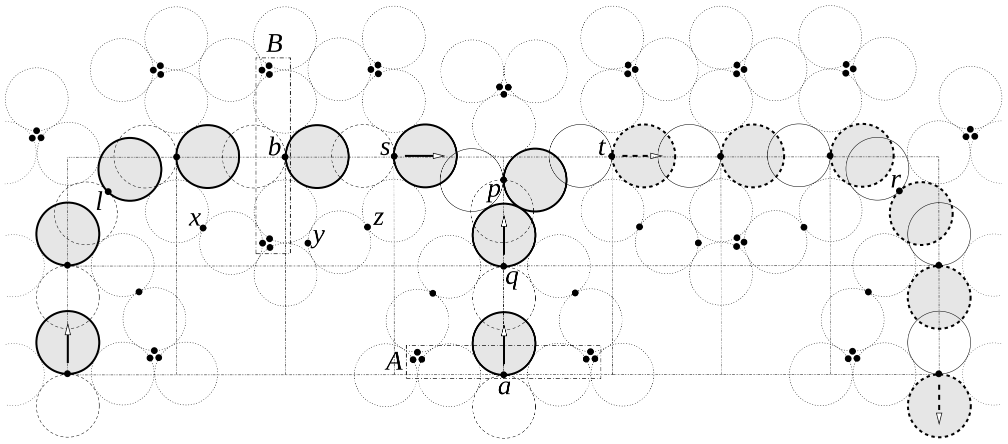

Figure 4.13: Clause with pressure from two variables. The current label positions are marked by shaded labels, alternatives are indicated by solid or dashed circles, and immobile labels are dotted.

We model the clauses by point sets that resemble large combs, see Figure 4.13. Such a comb consists of a horizontal part and three legs. The horizontal part is formed by active points like b in Figure 4.13 and by immobilizing clusters above and below b that restrict the label of b to two possible positions. All points of type b lie on a horizontal.

The legs consist of points like a in Figures 4.13 and 4.14. These points lie on a vertical line and are also forced into one of two possible positions. Where the legs are joined to the horizontal part of a comb, lack of space does not allow us to use the immobilizing clusters as elsewhere. Instead, we simply attach points like x, y, and z in Figure 4.13 to cluster labels in the vicinity such that the labels of x, y, and z are also immobile and at the same time restrict the label positions of active points (like l, r, s, t, and q) as desired.

Both the horizontal part and the legs of a comb can be extended as far as needed to reach the three variables belonging to the clause. This is done by repeating—at a distance of $\sqrt{3}$—patterns of seven points like those contained in the boxes A and B in Figure 4.13.

Each leg is connected to the encoding of a variable v. Let g be the vertical on which all points of the leg lie. It is perpendicular to the line h that contains all points encoding v. The lines g and h intersect in the midpoint d of one of the variable clusters on h, see Figure 4.14. The distance of $\sqrt{3}$ between the lowest leg point a and d is chosen to assure that the label of a can only intersect the label of d among the points modeling v. Note that the labels of a and d only intersect if the label of a is placed right below a and that of d right above d.

If variable v is negated in the clause under consideration, we join the leg to an odd cluster of v. Thus the cluster with d is labeled the same way as the leftmost cluster of v, see Figure 4.14. Now if v is set to true, d is labeled upwards. Then a and all other points on line g must also be labeled upwards. To put it graphically, pressure is transmitted upwards. If v is set to false, d can be labeled downwards, and no pressure is transmitted.

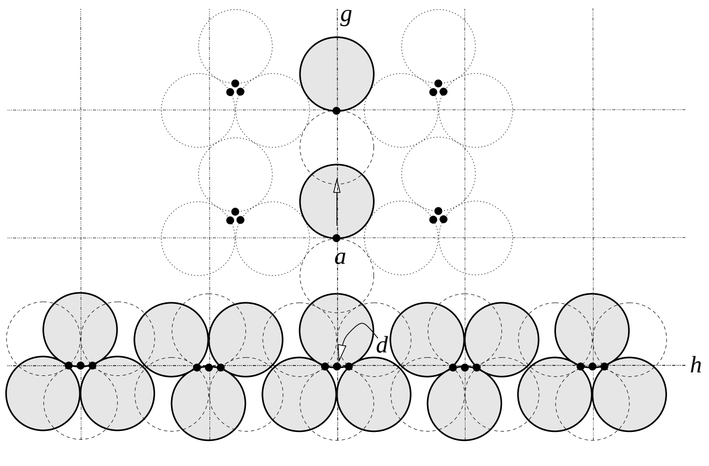

Figure 4.14: We connect the leg of a clause to a variable above the midpoint d of a variable cluster. If the variable is negated in the clause, then we join the leg to an odd cluster (as in this case), otherwise to an even cluster.

On the other hand, if v is not negated in the clause under consideration, then we join the clause's leg to an even cluster of v, which is labeled the opposite way as the leftmost cluster. Then pressure is transmitted if and only if v is set to false.

If all literals of a clause evaluate to false, then pressure is transmitted through all three legs into the clause. In this case there is a point (like p) that cannot be labeled, see Figure 4.13. In case there is at least one leg without pressure, it is obvious that all points belonging to a clause can be labeled. Hence the question whether ϕ can be satisfied is equivalent to asking whether all points in the set resulting from the encoding of ϕ can be labeled with unit circles.

To show that we can in fact connect a variable to several different clauses (below and above), we use a grid of width $\sqrt{3}$ and move all active points to grid points—except points of type l, r, and p, see Figure 4.13. In order to accommodate also the midpoints of the variable clusters on the grid (see Figure 4.14), we slightly increase the distance of neighboring variable clusters to that of immobilizing clusters, i.e. from $1 + \sqrt{2\sqrt{3} - 3} \approx 1.68$ to $\sqrt{3} \approx 1.73$. Then the label positions in every second cluster are still combinatorically equivalent, although labels can slightly wiggle now. Given such a grid embedding of our instance, it is clear that we can connect all variables to clauses according to ϕ.

We still have to ensure that the reduction is polynomial: if ϕ consists of m clauses and $n \leq 3m$ variables, the instance has $O(m^2)$ points. Their position can be computed in polynomial time if we round the grid-cell size and the distance between the three points of the immobilizing clusters to slightly greater rational numbers. The resulting instance is combinatorically equivalent to the one described before. ◶

Our gadget proof of the NP-hardness of circle labeling also shows that we cannot expect to approximate this problem arbitrarily well. Formann and Wagner used a similar argument to show that maximizing the size of axis-parallel square labels cannot be approximated beyond a factor of 1/2 [FW91]. In the formulation of their problem they only allow a constant number of label positions per point (namely four), which makes it easier to determine a good bound for the approximability.

Corollary 4.9 *There is a constant $\delta < 1$ such that it is NP-hard to label points with uniform circles of diameter greater than $\delta \cdot d_{\text{opt}}$.*

Proof. The proof of Theorem 4.8 still holds if the diameter of all labels is slightly reduced to a $\delta < 1$. The reason for this is that though labels have a certain degree of freedom now, every new label position is combinatorically equivalent to exactly one former position. δ must be chosen close enough to 1 to prevent a label from being moved continuously from one old position to another.

If there was a polynomial-time algorithm that could label the point set of the reduction with labels of size δ, we could solve planar 3-SAT in polynomial time and would thus have $\mathcal{P} = \mathcal{NP}$. ◶

The bottleneck that determines the minimum value of δ seems to be the encoding of a variable, see Figure 4.12. When the labels (and thus δ) are scaled down gradually, there is a point when two neighboring clusters can have identical instead of alternating label positions, see Figure 4.11. Then the variable's Boolean value is no longer well defined, and the proof collapses.