Optimal Algorithms for Approximate Clustering

Tomás Feder*
Computer Science Department
Stanford University
Stanford, CA 94305

Daniel H. Greene
Xerox Palo Alto Research Center
Palo Alto, CA 94304

Abstract

In a clustering problem, the aim is to partition a given set of n points in d-dimensional space into k groups, called clusters, so that points within each cluster are near each other. Two objective functions frequently used to measure the performance of a clustering algorithm are, for any L_q metric, (a) the maximum distance between pairs of points in the same cluster, and (b) the maximum distance between points in each cluster and a chosen cluster center; we refer to either measure as the cluster size.

We show that one cannot approximate the optimal cluster size for a fixed number of clusters within a factor close to 2 in polynomial time, for two or more dimensions, unless P=NP. We also present an algorithm that achieves this factor of 2 in time $O(n \log k)$, and show that this running time is optimal in the algebraic decision tree model. For a fixed cluster size, on the other hand, we give a polynomial time approximation scheme that estimates the optimal number of clusters under the second measure of cluster size within factors arbitrarily close to 1. Our approach is extended to provide approximation algorithms for the restricted centers, suppliers, and weighted suppliers problems that run in optimal $O(n \log k)$ time and achieve optimal or nearly optimal approximation bounds.

1. Introduction

Clustering problems arise in a great variety of contexts: data compression, pattern recognition, concept learning, point sampling, service site assignment, and windowing. Earlier work has shown that clustering is computationally hard in its optimal form, but in practice suboptimal solutions are very useful, and a variety of good heuristics are in widespread use (e.g. [Gr84], [Eq87]). The purpose of the present work is to explore suboptimal solutions, in particular, to find solutions that are guaranteed to be within a factor of the optimum, either in the number of clusters used, or in the size of clusters used.

Let $\mathbf{S}$ be a set of n points in d-dimensional space. A partition of $\mathbf{S}$ into k sets, $\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_k$, is called a *$k$-clustering*, and the individual $\mathbf{C}_i$ are called *clusters*. The *cluster size* of a k-clustering is the least value D for which all the points in each cluster $\mathbf{C}_i$ are either (a) within distance D of each other, or (b) within distance $D/2$ of some *cluster center* which can be any point in space, where distances are measured with any L_q metric. We refer to these two alternative definitions as *pairwise cluster size* and *central cluster size* respectively.

The *pairwise clustering problem* (*central clustering problem*) is to find, for a given set $\mathbf{S}$ and integer k, a k-clustering for $\mathbf{S}$ with pairwise cluster size (central cluster size) D as small as possible. (We will also consider variations where D is given and the smallest k is sought.) Notice that in the plane, with L_2 and L_∞ (or L_1) distances, the central clustering problem is the problem of covering the points in $\mathbf{S}$ with k circles or squares, respectively, of the smallest size.

*Work done in part as a summer intern at Xerox PARC, and in part at Stanford University with the support of an AT&T Bell Laboratories Scholarship.

In one dimension, pairwise and central clustering can be solved in polynomial time by dynamic programming ([Br77]); we can achieve $O(n \log n)$ running time. For two or more dimensions, the problem is known to be NP-complete (see Fowler, Paterson and Tanimoto [FPT81] for a reduction to central clustering).

Even if we allow the cluster size to be within a small factor α of the optimal size, the problem remains NP-hard. Megiddo and Supowit [MS84] showed NP-hardness results if $\alpha < 1.154$ for central L_2 clustering, and $\alpha < 1.5$ for central L_1 clustering. González [Go85] showed NP-hardness results for pairwise L_2 clustering when $\alpha < 1.732$.

We give a simpler reduction for these problems from planar vertex cover, which allows us to obtain better bounds for α. We show that the clustering problem remains NP-hard if $\alpha < 1.822$ for central L_2 clustering, $\alpha < 1.969$ for pairwise L_2 clustering, and $\alpha < 2$ for both pairwise and central L_1 and L_∞ clustering. Bounds of the form $\alpha < 2$ were previously known for a nongeometric version of the pairwise clustering problem, where the only constraint is that distances satisfy the triangle inequality [HS86], and thus without the geometric restrictions imposed by L_1 and L_∞ distances in the plane. For the geometric restrictions we consider, Mentzer has independently obtained the central clustering versions of the above bounds [Me88].

An approximation factor of 2 has in fact been achieved. Hochbaum and Shmoys ([HS86],[HS85]) gave a graph algorithm for the pairwise clustering problem which runs in time $O(n^2 \log n)$. Independently, González [Go85] achieved the same approximation factor in $O(kn)$ time. One of our main results is to show that for points in d-dimensional space and under any L_q metric, a solution which achieves an approximation factor of $\alpha = 2$ for both central and pairwise clustering can be found in $O(n \log k)$ time. We also show a matching lower bound of $\Omega(n \log k)$ under the algebraic decision tree model, so our algorithm has essentially the best possible running time.

The $O(n \log k)$ algorithm mentioned above uses an extension of the box decomposition method of Vaidya [Va86], a technique that also applies to a variety of problems closely related to clustering. Four problems which can be solved approximately in polynomial time are the *restricted centers* and the *suppliers* problems, as well as their weighted counterparts, the *weighted centers* and *weighted suppliers* problems (see [HS86]). We achieve, in d-dimensional space, the best known approximation factors α in optimal $O(n \log k)$ time: the factors are $\alpha = 2$ for the restricted centers problem, and $\alpha = 3$ for the three other problems. For $d \geq 2$ we show that restricted centers becomes NP-hard if $\alpha < 1.732$ for the L_2 metric and $\alpha < 2$ for the L_1 and L_∞ metrics (Mentzer has independently achieved these two bounds and recently improved the L_2 bound to 1.932 [Me88]), and the suppliers problem becomes NP-hard if $\alpha < 2.645$ for the L_2 metric and $\alpha < 3$ for the L_1 and L_∞ metrics. These bounds apply to the weighted versions as well. (Bounds for the values of α for which these problems remain NP-hard were previously known only under the general triangle inequality constraint mentioned earlier. Here Hsu and Nemhauser [HN79] showed that restricted centers is NP-hard for $\alpha < 2$, while H. Karloff (see [HS86]) showed that the suppliers problem remains NP-hard for $\alpha < 3$.)

Another way of seeking approximate solutions is to fix a cluster size D, say $D = 1$, and then find solutions with as few clusters as possible. We show that it is possible to obtain, in polynomial time, solutions for the central clustering problem that achieve any fixed approximation factor $\beta > 1$ for the number of clusters k. Our algorithm runs in time $O(c(\rho, l)l^2 n + n \log n)$, for $d = 2$ and metric L_∞, where ρ is the largest number of points of $\mathbf{S}$ contained in a single unit square, $c(\rho, l) = 4^{l-1}(2\rho)^{2(2l-1)}$, and $\beta = 1 + 1/l$ is the quality of the approximation. When β is as large as 2, the running time can be improved to $O(n \log k)$. These results generalize easily to higher dimensions, and can be further improved by tolerating some approximation factor $\alpha > 1$ for the cluster size.

The remainder of this paper is organized as follows. Section 2 derives the NP-hardness results. Section 3 proves the lower bounds on the running time of approximation algorithms. Section 4 describes the $O(n \log k)$ algorithm which approximates the cluster size with a performance bound of 2, and section 5 implements the box decomposition scheme used by the algorithm. Section 6 presents the dynamic programming algorithm which approximates the optimal number of clusters. Section 7 explores various extensions of the results of the previous sections.

<==<GraphTrans.ip<

Figure 1. Transforming a graph

<==<Cons.ip<

Figure 2. Detail of a node with three edges for (a) central clustering, and (b) pairwise clustering.

2. NP-hardness of Approximate Clustering

An instance of vertex cover for planar graphs of degree at most 3 ([GJ77]) can be embedded in the plane, by replacing all edges with odd length paths, so that edges become segments of length 1 [see Fig. 1]. The midpoints of these edges then form an instance of central clustering which has a k-clustering with cluster size 1 if and only if the embedded graph has a vertex cover with k nodes. We can use this construction to show that approximate clustering is NP-complete:

Theorem 1 *Approximate clustering in $d \geq 2$ dimensions is NP-hard for approximation factors of α in the cluster size if $\alpha < (1+\sqrt{7})/2 \approx 1.82$ for central L_2 clustering, $\alpha < 2\cos(10ffo) \approx 1.97$ for pairwise L_2 clustering, and $\alpha < 2$ for pairwise and central L_1 and L_∞ clustering.*

Proof. First note that, for central clustering, we can guarantee that the midpoints $m(e)$ and $m(e')$ of two edges e and e' in the graph are at distance at least $d_1 = \sqrt{13}/2 \approx 1.80$ From each other unless e and e' have one of their endpoints in common [see Fig. 2a]. Thus finding an approximate solution within an approximation factor smaller than d_1 is no easier than finding an exact solution. A similar argument applies to pairwise clustering, with the critical distance $d_2 = (1+\sqrt{3})/\sqrt{2} \approx 1.93$ shown in Fig. 2b. The better bounds of $d_1' \approx 1.82$ and $d_2' \approx 1.97$ in the statement of the theorem are obtained by replacing the constructions in Fig. 2 with the corresponding ones shown in Fig. 3. To see that the old and new constructions are in fact equivalent, call the three outermost points in each of these figures *external*, and the remaining points *internal*. Then all the constructions have the property that if the minimum of three clusters of size 1 is used to cover the internal points then none of the external points can be covered, but if four clusters are used then the external points can be covered as well. Furthermore this behavior holds in the refined constructions for cluster sizes as large as the bounds given in the theorem.

The bound of $d = 2$ for L_∞ distances is easier to obtain and is illustrated in Fig. 4; for L_1 distances, we simply rotate this figure by 45°. □

<==<AdvCons.ip<

Figure 3. Refined construction for (a) central clustering, and (b) pairwise clustering.

3. Lower Bounds on the Time Complexity

We begin with an "information theoretic" lower bound. For each possible partition of n elements into k nonempty sets, there is an instance of the problem in which the k-clustering defined by that partition is

<==<Linf.ip<

Figure 4. Construction for L_∞ metric

the only one close to optimal. Since there are at least k^{n-k} possible partitions, we have:

Theorem 2 *Finding a k-clustering with cluster size within any given factor α of the optimum requires $\Omega((n-k)\log k)$ time under the decision tree model, for both pairwise and central clustering, in $d \geq 1$ dimensions and under any L_q metric.*

Information theoretic arguments count the possible outcomes of an algorithm, and thus cannot provide a good lower bound if the number of such outcomes is small. This is the case in our problem when k is very close to n. On the other hand, if we restrict the set of allowed operations by adopting the algebraic decision tree model ([Be83],[PS85]), we can obtain a lower bound which holds even when there are only two possible outcomes, which is the case for the following decision problem. We say that an algorithm solves the *approximate clustering decision problem* with an approximation factor of α if, given a set of n points, the algorithm answers the question "Is there a k-clustering with cluster size αD_0 or less?" as follows: (a) If the cluster size of the optimal k-clustering is $D > \alpha D_0$, the algorithm answers NO, since then no such clustering exists; (b) If $D \leq D_0$, the algorithm answers YES, because then any k-clustering within a factor of α of the optimum has cluster size $\alpha D \Delta 0$ or less; and (c) For intermediate values of D, that is, $D_0 < D \leq \alpha D_0$, the algorithm may answer either YES or (incorrectly) NO. We show:

Theorem 3 *The approximate clustering decision problem requires $\Omega(n \log k)$ time for $k < n$ in d-dimensional space under the algebraic decision tree model, for both pairwise and central clustering, any given L_q distance metric, and $d \geq 1$.*

Proof. Without loss of generality, we can restrict the problem to one dimension and let $D_0 = 1$. An *instance* of the problem is then an n-tuple $I = (x_1, \ldots, x_n) \in \mathbf{R}^n$ of n points on the real line. Let $\mathbf{W}$ be the set of instances for which an algorithm solving the approximate clustering decision problem answers YES. We first restrict the problem to the subspace $\mathbf{U}$ of $\mathbf{R}^n$ in which the first k points $\mathbf{T} = \{x_1, \ldots, x_k\}$ are fixed points at distance greater than 2α from each other. We want to show that $\mathbf{W} \cap \mathbf{U}$ has many components. Notice that we can choose a *neighbor* from $\mathbf{T}$ for each x_i among the remaining $n-k$ points $x_{k+1}, \ldots, x_n$, in k^{n-k} possible ways; then for each such possible *neighbor assignment*, we can define an instance I in which each x_i among $x_{k+1}, \ldots, x_n$ is within distance $1/2$ of its assigned neighbor. This instance I has a k-clustering with cluster size 1, and so $I \in \mathbf{W} \cap \mathbf{U}$. Now consider two instances I_1 and I_2 corresponding to two different neighbor assignments. There must be some x_i among $x_{k+1}, \ldots, x_n$ which has a neighbor in I_2 different from its neighbor in I_1. Since the points $\mathbf{T}$ are at distance greater than 2α from each other, any continuous path in $\mathbf{U}$ from I_1 to I_2 must pass through an instance I_3 in which x_i is at distance greater than α From all points in $\mathbf{T}$. Then all the points in $\mathbf{T} \cup \{x_i\}$ are at distance greater than α from each other, so I_3 has no k-clustering with cluster size α or less, therefore $I_3 \in \overline{\mathbf{W}} \cap \mathbf{U}$. It follows that I_1 and I_2 are in different components of $\mathbf{W} \cap \mathbf{U}$, so $\mathbf{W} \cap \mathbf{U}$ has $\#(\mathbf{W} \cap \mathbf{U}) \geq k^{n-k}$ components (at least one for each neighbor assignment), and therefore $\log \#(\mathbf{W} \cap \mathbf{U}) \geq (n-k)\log k$, providing an $\Omega((n-k)\log k)$ lower bound.

Now we restrict the problem to the subspace $\mathbf{U}'$ orthogonal to $\mathbf{U}$ in which the last $n-k$ points $\mathbf{T}' = \{x_{k+1}, \ldots, x_n\}$ are fixed points within distance 1 of each other. This time we want to show that $\overline{\mathbf{W}} \cap \mathbf{U}'$ has many components. For each choice out of the $k!$ possible orderings for $x_1, \ldots, x_k$, we define an instance I' in which these k points are in the chosen order on the real line, at distance greater than α from each other and from $\mathbf{T}'$. This instance I' has no k-clustering with cluster size α or less, so $I' \in \overline{\mathbf{W}} \cap \mathbf{U}'$. Consider now two instances I_1' and I_2' corresponding to two different orderings. There must exist two points among $x_1, \ldots, x_k$, say x_1 and x_2, which are in different order in I_1' and in I_2'. Then any continuous path in $\mathbf{U}'$ From I_1' to I_2' must pass through an instance I_3' in which x_1 and x_2 are within distance 1 of each other. Then I_3' has a k-clustering with clusters $\{x_1, x_2\}$,$\{x_3\}$, …,$\{x_k\}$,$\mathbf{T}'$ and cluster size at most 1, so $I_3' \in \mathbf{W} \cap \mathbf{U}'$. It follows that I_1' and I_2' are in different components of $\overline{\mathbf{W}} \cap \mathbf{U}'$, so $\overline{\mathbf{W}} \cap \mathbf{U}'$ has $\#(\overline{\mathbf{W}} \cap \mathbf{U}') \geq k!$ components (at least one for each ordering). Taking logarithms, we obtain an $\Omega(k \log k)$ lower bound, which combined with the previous bound gives us an $\Omega(n \log k)$ lower bound. □

It is interesting to observe how, by restricting the problem to the two orthogonal subspaces $\mathbf{U}$ and $\mathbf{U}'$, it is possible to obtain complementary bounds on the number of components for $\mathbf{W}$ and $\overline{\mathbf{W}}$ respectively.

The $O(n \log k)$ algorithm of the next section belongs to the algebraic decision tree model (as long as the L_q metric has q rational or infinite), so the lower bound given by Theorem 3.2 is in fact a tight bound.

4. Approximating the Cluster Size

A standard paradigm for devising approximation algorithms is to find a set that is easy to compute and then make double use of this set: both to prove a lower bound on the optimal solution, and to produce an approximate solution. González uses such a strategy to solve the approximate clustering problem. The "farthest point" set $\mathbf{T}$ is easy to compute, bounds the optimal solution, and provides clusters for an approximate solution. For each point not in $\mathbf{T}$, the algorithm keeps *neighbor*(p), the nearest point in $\mathbf{T}$, and *dist*(p), the distance from p to *neighbor*(p):

Farthest Point Algorithm

$\mathbf{T} \leftarrow \emptyset$;
dist$(p) \leftarrow \infty$ for all $p \in \mathbf{S}$;
while$|\mathbf{T}| \leq k$ **do**
$D \leftarrow \max\{dist(p) \mid p \in \mathbf{S} - \mathbf{T}\}$;
choose $\hat{p}$ from $\mathbf{S} - \mathbf{T}$ such that $dist(\hat{p}) = D$;
add $\hat{p}$ to $\mathbf{T}$;
update *neighbor*(p) and *dist*(p) for all $p \in \mathbf{S} - \mathbf{T}$;
endloop.

At the end of the algorithm there are $k+1$ points at distance at least D From each other, and the optimal solution must have at least two of these points in the same cluster, so the cluster size is at least D. However, at the beginning of the last iteration, after the last assignment to D, but before the last choice of $\hat{p}$, we have k points in $\mathbf{T}$ that can be used to define clusters. For each $p_i \in \mathbf{T}$ we define a corresponding $\mathbf{C}_i$ that includes all nearby points p for which *neighbor*$(p) = p_i$. Since *dist*$(p) \leq D$ for all p, this clustering has size at most $2D$, which is within a factor of 2 of the optimum, for both pairwise and central clustering. The running time depends principally on the update step at the end of the loop; it takes $O(n)$ time per iteration of the loop, for a total running time of $O(kn)$.

The purpose of this section is to improve the running time of the above algorithm to match the lower bound of the preceding section. We will not need to compute the same set of points computed above, but only to satisfy the ultimate goal of producing an approximate solution within a factor of 2. Our algorithm will run in two phases. Both phases will resemble the farthest point algorithm, but the first phase will operate on points grouped together in boxes, and produce a crude solution within a factor of 6. Using the crude bound and the boxes derived in the first phase, the second phase will run a modification of the above algorithm where individual points will be processed, but the updates will be limited to nearby boxes, thus achieving the desired efficiency.

It is convenient to view the farthest point algorithm as an independent set computation. Initially we form a graph $\mathbf{G} = (\mathbf{S}, \mathbf{E})$, where the nodes are the points of the original problem, and the edges form a complete graph, with weights equal to the distances between endpoints. The algorithm computes an independent set $\mathbf{T}$, by alternating between adding points of $\mathbf{S}$ to $\mathbf{T}$ until every point of $\mathbf{S} - \mathbf{T}$ is adjacent to a point in $\mathbf{T}$, and deleting a longest edge of $\mathbf{G}$. This computes the same set $\mathbf{T}$ as the farthest point algorithm above. While at first glance the independent set computation appears to be considerably less efficient than the farthest point algorithm, it is the most natural way to describe the first phase of our approximation algorithm.

The first phase groups the points in $\mathbf{S}$ into rectangular boxes and maintains a graph $\mathbf{G}$ between *boxes* rather than *points*. Central to its performance is the efficient implementation of an extension, described below, of Vaidya's box decomposition method [Va86].

Let a box b in d-dimensional space be the product $J_1 \times J_2 \times \cdots \times J_d$ of d intervals. Given a box b, we denote by *shrunk*(b) the smallest box containing all the points in $b \cap \mathbf{S}$, and by *box*$(\mathbf{S})$ the smallest box containing all the points in $\mathbf{S}$. The box decomposition scheme maintains a set of boxes $\mathbf{B}$ containing the points in $\mathbf{S}$, splitting the boxes until some condition is satisfied:

Box Decomposition Scheme

$\mathbf{B} \leftarrow \{box(\mathbf{S})\}$;
repeat
pick a box b from $\mathbf{B}$;
choose disjoint boxes b_1 and b_2 such that $b = b_1 \cup b_2$;
$\mathbf{B} \leftarrow (\mathbf{B} - \{b\}) \cup \{shrunk(b_1), shrunk(b_2)\}$;
untilsatisfied;
$k \leftarrow |\mathbf{B}|$.

In the next section we will show that this algorithm can be implemented in time $O(n \log k)$. For our clustering algorithm, we will always split boxes b with a hyperplane $h(b)$ passing through the center of b and perpendicular to a *longest dimension* of b, that is, a dimension i for which the interval J_i in the definition of b is largest. We denote by $length(b)$ the length of a largest J_i. If the hyperplane $h(b)$ splits box b into boxes b_1 and b_2, we call the boxes $shrunk(b_1)$ and $shrunk(b_2)$ the *successors* of b.

The algorithm for the first phase is given below. For a given graph $\mathbf{G} = (\mathbf{B}, \mathbf{E})$ on the set of boxes $\mathbf{B}$, if $e = (b_1, b_2)$ is an edge in $\mathbf{E}$, we denote by $length(e)$ the minimum distance between the boundaries of b_1 and b_2. We maintain a threshold D' such that at each iteration, after the assignment to D' is executed, and also at the end of the loop, the following invariants hold: (a) All boxes $b \in \mathbf{B}$ satisfy $length(b) \le D'/d$ (and hence no two points in the same box are further apart than D'); (b) The graphs $\mathbf{G}_{D'} = (\mathbf{B}, \mathbf{E}_{D'})$ and $\mathbf{G}_{3D'} = (\mathbf{B}, \mathbf{E}_{3D'})$ include those edges e for which $length(e) < D'$ and $length(e) < 3D'$ respectively, and exclude those edges for which $length(e) > D'$ and $length(e) > 3D'$ respectively; and (c) The set $\mathbf{T}'$ is a maximal independent set in $\mathbf{G}_{D'}$. In the algorithm below the computations of $\mathbf{G}_{D'}$ and $\mathbf{G}_{3D'}$ are identical. The graph $\mathbf{G}_{D'}$ is used immediately in the first phase to construct a maximal independent set $\mathbf{T}'$, while the graph $\mathbf{G}_{3D'}$ will not be needed until the second phase.

First Clustering Phase

$\mathbf{B} \leftarrow \{box(\mathbf{S})\}$; $\mathbf{T}' \leftarrow \mathbf{B}$;
$\mathbf{G}_{D'} \leftarrow (\mathbf{B}, \emptyset)$; $\mathbf{G}_{3D'} \leftarrow (\mathbf{B}, \emptyset)$;
while$|\mathbf{T}'| \le k$ **do**
$D' \leftarrow \min\{D' \mid length(b) \le D'/d$,
$length(e_1) \le D'$, and $length(e_3) \le 3D'$,
for all $b \in \mathbf{B}$, $e_1 \in \mathbf{E}_{D'}$, and $e_3 \in \mathbf{E}_{3D'}\}$;
if$length(\hat{b}) = D'/d$ for some box $\hat{b} \in \mathbf{B}$ **then**
replace $\hat{b}$ in $\mathbf{B}$ by its successors b_1^* and b_2^*,
remove $\hat{b}$ from $\mathbf{T}'$ if necessary, and replace
all edges $e = (\hat{b}, b)$ in $\mathbf{G}_{D'}$ and $\mathbf{G}_{3D'}$ by
corresponding edges $e_1^* = (b_1^*, b)$, $e_2^* = (b_2^*, b)$;
remove from $\mathbf{G}_{D'}$ and $\mathbf{G}_{3D'}$ all edges e for which
$length(e) \ge D'$ and $length(e) \ge 3D'$
respectively, adding to $\mathbf{T}'$ any boxes $b \in \mathbf{B} - \mathbf{T}'$
not adjacent in $\mathbf{G}_{D'}$ to a box in $\mathbf{T}'$;
endloop.

At the end of the algorithm, $\mathbf{T}'$ has $k+1$ or more boxes at least D' apart, giving a lower bound of D' for the optimal cluster size. On the other hand, immediately after the last assignment to D', every box in $\mathbf{B}$ is adjacent in $\mathbf{G}_{D'}$ to some box in the maximal independent set $\mathbf{T}'$, which has no more than k boxes; so if we now choose a center p_i from each box $b_i \in \mathbf{T}'$, then every point in $\mathbf{S}$ is within distance $3D'$ of some p_i [see Fig. 5]. This gives the following:

Lemma 1 *The first clustering phase produces a k-clustering with cluster size within a factor of 6 of the optimum, for both pairwise and central clustering.*

<==<SixD.ip<

Figure 5. Solution with cluster size $6D'$

The main property which allows us to implement this algorithm efficiently is the fact that the graph $\mathbf{G}_{\lambda D} = (\mathbf{B}, \mathbf{E}_{\lambda D})$, where $\mathbf{E}_{\lambda D}$ consists of the edges e for which $length(e) \le \lambda D$, has bounded degree. This follows from a packing argument as in [Va86].

Lemma 2 *Every node in $\mathbf{G}_{\lambda D}$ has degree at most $2ffd(2\lambda d+3)ffd$ during the entire execution of the algorithm.*

Lemma 3 *The time complexity of the first clustering phase is $O(n \log k)$ for the box decomposition and $O(n + k \log k)$ for the remaining steps.*

PROOF. The running time of the first phase is dominated by the box decomposition, which we will show takes $O(n \log k)$ time in the next section. The remaining time can be seen to be $O(n + k \log k)$ as follows. At the beginning of the last iteration of the while loop, the set $\mathbf{T}'$ contains at most k boxes, so by the preceding lemma, the number of boxes adjacent in $\mathbf{G}_{D'}$ to boxes in $\mathbf{T}'$ is $O(k)$, and these are all the boxes in $\mathbf{B}$, since $\mathbf{T}'$ is a maximal independent set. Moreover, since the number of boxes in $\mathbf{B}$ increases by 1 whenever two new boxes are created to replace an old box in $\mathbf{B}$, the total number of boxes generated by the algorithm is also $O(k)$. By the preceding lemma, only $O(1)$ edges are generated when a new box is created, so the total number of edges generated for $\mathbf{G}_{D'}$ and $\mathbf{G}_{3D'}$ is also $O(k)$. It follows that the number of operations performed in the while loop is $O(k)$. In order to find the longest box in $\mathbf{B}$ and the longest edges in $\mathbf{G}_{D'}$ and $\mathbf{G}_{3D'}$, we maintain all boxes and edges in a priority queue, which can be updated in time $O(\log k)$ per operation, for a total $O(k \log k)$ time. The k-clustering can be obtained at the end of the phase in $O(n)$ time by visiting each box $b \in \mathbf{T}'$ and boxes adjacent to b in $\mathbf{G}_{D'}$. $\square$

In the second phase, we construct a k-clustering with cluster size within a factor of 2 of the optimum. All we need from the first phase is the value of the threshold D' and the graph $\mathbf{G}_{3D'} = (\mathbf{B}, \mathbf{E}_{3D'})$. The algorithm is similar to the farthest point algorithm presented at the beginning of the section with the exception that we treat distances greater than $3D'$ as infinite. If a point p is at distance $3D'$ or more from a point in $\mathbf{T}''$ then *neighbor*(p) is undefined and *dist*$(p) = \infty$. As a consequence, we need only search nearby boxes during the update step at the end of the loop:

Second Clustering Phase
$\mathbf{T}'' \leftarrow \emptyset$; $D'' \leftarrow \infty$;
dist$(p) \leftarrow \infty$ for all $p \in \mathbf{S}$;
while$|\mathbf{T}''| \leq k$ and $D'' > D'$ **do**
$D'' \leftarrow \max\{$*dist*$(p) \mid p \in \mathbf{S} - \mathbf{T}''\}$;
choose $\hat{p}$ from $\mathbf{S} - \mathbf{T}''$ such that *dist*$(\hat{p}) = D''$;
add $\hat{p}$ to $\mathbf{T}''$;
let $\hat{b}$ be the box containing $\hat{p}$;
update *neighbor*(p) and *dist*(p) for all $p \in \mathbf{S}$
in boxes adjacent to $\hat{b}$ in $\mathbf{G}_{3D'}$;
endloop.

If $D'' = \infty$ at the end of the algorithm, then $\mathbf{T}''$ has $k+1$ points at least $3D'$ apart, so the k-clustering from the first phase, with cluster size no greater than $6D'$, has size at most twice the optimum. Otherwise, after the last assignment to D'', the points in $\mathbf{T}''$ define a k-clustering with cluster size at most $2D''$ as in the farthest point algorithm, and we have a lower bound of D'' because at the end either $\mathbf{T}''$ has $k+1$ points at least D'' apart, or D'' is no greater than the lower bound of D' from the first phase. This gives:

Lemma 4 *The second clustering phase produces a k-clustering with cluster size within a factor of 2 of the optimum, for both pairwise and central clustering.*

Lemma 5 *The time complexity of the second clustering phase is $O(n + k \log k)$.*

PROOF. We maintain, for each box $b \in \mathbf{B}$, a candidate point *candidate*(b): this is a point $p \in b \cap \mathbf{S}$ for which *dist*(p) is largest. We can then always pick the next point to be added to $\mathbf{T}''$ from the set of candidate points. Notice that at most one point per box can be added to $\mathbf{T}''$ up to the last assignment to D'', since points in the same box are within distance D' of each other. Also, when a point $\hat{p} =$ *candidate*$(\hat{b})$ is added to $\mathbf{T}''$, we only need to update *candidate*(b), *neighbor*(p) and *dist*(p) for boxes $b \in \mathbf{B}$ and points $p \in b \cap \mathbf{S}$ such that $\hat{b}$ and b are adjacent in $\mathbf{G}_{3D'}$, since distances greater than $3D'$ are treated as ∞. Therefore, by Lemma 4.2, each such box b and point p is updated $O(1)$ times. We thus need only $O(n)$ time to perform these updates. To determine a candidate point $\hat{p}$ for which *dist*$(\hat{p})$ is largest, we maintain the candidate points in a priority queue. We only update this queue $O(k)$ times, since there are $O(k)$ boxes and the candidate point for each box is updated $O(1)$ times. Therefore the queue operations take time $O(k \log k)$. Finally, the k-clustering can be constructed in $O(n)$ time, since it only requires grouping together all points $p \in \mathbf{S}$ which have the same *neighbor*(p). $\square$

Combining both phases we obtain, by the last three lemmas:

Theorem 4 *The clustering algorithm finds a k-clustering with cluster size within a factor of 2 of the optimum in $O(n \log k)$ time, for both central and pairwise clustering, in d-dimensional space, and under any given L_q distance metric.*

5. Implementation of Box Decomposition

This section shows how to implement the box decomposition scheme introduced in the preceding section in $O(n \log k)$ time. It would be useful to have the points in each box sorted along each dimension as in [Va86]; sorting, however, requires $\Omega(n \log n)$ time. We therefore keep the points only partially sorted. More precisely, for each box $b \in \mathbf{B}$ and each dimension $1 \leq i \leq d$, we partition the points in $b \cap \mathbf{S}$ into disjoint nonempty sets and keep these sets in a sorted list $L_i(b)$, so that if point $x = (x_1, \ldots, x_d)$ belongs to the set A_x, point $y = (y_1, \ldots, y_d)$ belongs to the set A_y, and A_x occurs to the left of A_y in $L_i(b)$, then $x_i \leq y \Delta i$.

Initially, a single box $b = box(\mathbf{S})$ contains all the points in $\mathbf{S}$. The points are not sorted, so each $L_i(b)$ consists of a single set with all the points in $\mathbf{S}$. We then repeat the following steps $k-1$ times. At each iteration, we pick a box b from the current set of boxes $\mathbf{B}$ and a hyperplane $x_i = M$, and we split the points $x \in b$ into two boxes depending on whether $x_i \leq M$ or $x_i > M$. We call the box containing fewer points b_{small}, and the other box b_{large}, so that $|b_{\mathrm{small}} \cap \mathbf{S}| \leq |b_{\mathrm{large}} \cap \mathbf{S}|$. Finally, we determine the shrunk boxes $shrunk(b_1)$ and $shrunk(b_2)$. This splitting and shrinking is done as follows:

1. Traverse $L_i(b)$ simultaneously from the left and right ends, until all the points in $b_{\mathrm{small}} \cap \mathbf{S}$ have been found. Now do the following steps separately for each coordinate $1 \leq i \leq d$.

2. Extract from each set in $L_i(b)$ the points in $b_{\mathrm{small}} \cap \mathbf{S}$, so that each set in $L_i(b)$ is split into a set containing only points in $b_{\mathrm{small}} \cap \mathbf{S}$ and a set containing only points in $b_{\mathrm{large}} \cap \mathbf{S}$. Then traverse $L_i(b)$ from left to right, building two lists $L_i(b_{\mathrm{small}})$ and $L_i(b_{\mathrm{large}})$ consisting of the nonempty sets of points in $b_{\mathrm{small}} \cap \mathbf{S}$ and $b_{\mathrm{large}} \cap \mathbf{S}$ respectively, in the order they occur in $L_i(b)$.

3. Visit the leftmost and rightmost sets of both lists $L_i(b_{\mathrm{small}})$ and $L_i(b_{\mathrm{large}})$ to determine the boundaries of $shrunk(b_1)$ and $shrunk(b_2)$. For each set $\mathbf{A}$ visited, if $|\mathbf{A}| \geq 2$, find the median of the x_i coordinates of the points $x \in \mathbf{A}$ and split $\mathbf{A}$ into two sets $\mathbf{A}_L$ and $\mathbf{A}_R$ with $\lfloor |\mathbf{A}|/2 \rfloor$ and $\lceil |\mathbf{A}|/2 \rceil$ points respectively, so that $x_i \leq y_i$ for all points $x \in \mathbf{A}_L$ and $y \in \mathbf{A}_R$. Then replace the set $\mathbf{A}$ in the list by the two adjacent sets $\mathbf{A}_L$ and $\mathbf{A}_R$.

To analyze this algorithm, we prove a combinatorial lemma for a family of sets that have the special property that similarly sized sets are disjoint:

Lemma 6 *Suppose $\mathbf{E}_0, \ldots, \mathbf{E}_{k-1}$ are k subsets of a set of n elements such that if $\lceil |\mathbf{E}_{j'}|/2 \rceil < |\mathbf{E}_j| \leq |\mathbf{E}_{j'}|$ then $\mathbf{E}_j$ and $\mathbf{E}_{j'}$ are disjoint. Then $\sum_j |\mathbf{E}_j|$ is $O(n \log k)$.*

Proof. There are fewer than 2^{s+1} sets $\mathbf{E}_j$ which satisfy $\lceil n/2^{s+1} \rceil < |\mathbf{E}_j| \leq \lceil n/2^s \rceil$, since all such E_j must be disjoint. Therefore the sum is at most $\sum_{s=0}^{\lfloor \log k \rfloor} 2^{s+1} \lceil n/2^s \rceil = O(n \log k)$. □

We analyze the running time of this algorithm separately for each dimension $1 \leq i \leq d$. Let $\mathbf{S}_j$ be the set of points $b_{\mathrm{small}} \cap \mathbf{S}$ found in step 1 during the jth iteration, and let $\mathbf{A}_{4j}, \mathbf{A}_{4j+1}, \mathbf{A}_{4j+2}, \mathbf{A}_{4j+3}$, be the four sets visited in step 3 during the jth iteration (for $j = 0, 1, \ldots, k-2$).

Lemma 7 *The sums $\sum_j |\mathbf{S}_j|$ and $\sum_j |\mathbf{A}_j|$ are both $O(n \log k)$.*

Proof. When a box b is split into two boxes b_{small} and b_{large}, we always have $|b_{\mathrm{small}} \cap \mathbf{S}| \leq \lceil |b \cap \mathbf{S}|/2 \rceil$, so $\lceil |\mathbf{S}_{j'}|/2 \rceil < |\mathbf{S}_j| \leq |\mathbf{S}_{j'}|$ cannot hold unless $\mathbf{S}_j$ and $\mathbf{S}_{j'}$ are disjoint. By the preceding lemma, the sum of the $|\mathbf{S}_j|$ is $O(n \log k)$. Similarly, when a set $\mathbf{A}$ is split into two sets $\mathbf{A}_L$ and $\mathbf{A}_R$ in step 3 of the algorithm, we have $|\mathbf{A}_L|, |\mathbf{A}_R| \leq \lceil |\mathbf{A}|/2 \rceil$, so $\lceil |\mathbf{A}_{j'}|/2 \rceil < |\mathbf{A}_j| \leq |\mathbf{A}_{j'}|$ cannot hold unless $\mathbf{A}_j$ and $\mathbf{A}_{j'}$ are disjoint, and the preceding lemma implies that the sum of the $|\mathbf{A}_j|$ is also $O(n \log k)$. □

Lemma 8 *The length of the lists $L_i(b)$ is at most $\min(2k, 2 \log n + 3)$.*

Proof. The length of a list $L_i(b)$ only increases when the leftmost or the rightmost set in $L_i(b)$ is split during step 3 of the algorithm, so it increases by at most 2 at each iteration of the algorithm. Since the initial list length is 1 and there are $k-1$ iterations, the list length cannot exceed $2k$.

We now claim that in every list $L_i(b)$ there is a *central set* $\mathbf{A}$ such that for every set $\mathbf{A}'$ in $L_i(b)$, if $\mathbf{A}'$ is at distance j from $\mathbf{A}$ in $L_i(b)$, then $|\mathbf{A}'| \leq \lceil n/2^j \rceil$. This claim implies the lemma, because a set $\mathbf{A}'$ at distance $\lceil \log n \rceil$ from $\mathbf{A}$ in $L_i(b)$ will never be split during step 3 of the algorithm because $|\mathbf{A}'| = 1$, so all sets in $L_i(b)$ are within distance $\lceil \log n \rceil$ of $\mathbf{A}$, giving a total list length of at most $2\lceil \log n \rceil + 1$.

To prove the claim, notice that the property holds at the beginning of the algorithm, when all lists have length 1. Step 2 preserves the property because when we split $L_i(b)$ into two lists $L_i(b_{\text{small}})$ and $L_i(b_{\text{large}})$ we only decrease the number of points in each set of the list, and discarding empty sets only brings sets in the list closer to the central set $\mathbf{A}$. Step 3 also preserves the property because when a rigthmost (or leftmost) set $\mathbf{A}'$ at distance j from the central set $\mathbf{A}$ is split, creating a new set $\mathbf{A}''$ at distance $j+1$ from $\mathbf{A}$, we will have $|\mathbf{A}''| \le \lceil n/2^{j+1} \rceil$ since $|\mathbf{A}''| \le \lceil |\mathbf{A}'|/2 \rceil$ and $|\mathbf{A}'| \le \lceil n/2^j \rceil$. Since step 1 does not modify the lists, this proves the claim and completes the proof of the lemma. □

Theorem 5 *The box decomposition algorithm runs in time $O(n \log k)$.*

Proof. In step 1, during the jth iteration, one of the two equal-sized searches only visits points in $b_{\text{small}} \cap \mathbf{S}$ plus some extra points in $b_{\text{large}} \cap \mathbf{S}$ which will end up, after the splitting, in either the leftmost or the rightmost set of $L_i(b_{\text{large}})$. Therefore all these points are in $\mathbf{S}_j \cup \mathbf{A}_{4j} \cup \mathbf{A}_{4j+1} \cup \mathbf{A}_{4j+2} \cup \mathbf{A}_{4j+3}$ and step 1 takes $O(n \log k)$ time by Lemma 5.2. In step 2 we only visit points in $\mathbf{S}_j = b_{\text{small}} \cap \mathbf{S}$, and this again takes $O(n \log k)$ time. The time spent traversing $L_i(b)$ is $O(\min(k, \log n))$ by Lemma 5.3, giving over the $k-1$ iterations a total $O(k \min(k, \log n)) = O(n \log k)$ time. Finally, only points in $\mathbf{A}_{4j} \cup \mathbf{A}_{4j+1} \cup \mathbf{A}_{4j+2} \cup \mathbf{A}_{4j+3}$ are visited in step 3 during the jth iteration, so by Lemma 5.2 this step only requires $O(n \log k)$ time as well. □

6. Approximating the Number of Clusters

We will now consider the alternate problem: given a set $\mathbf{S}$ and real D, find the smallest k such that $\mathbf{S}$ has a central k-clustering of size D. We can approximate the minimum number of clusters by dividing the clustering problem into strips of fixed width, and then solving the problem exactly on each strip. First we will show that a horizontal strip H_l of fixed height lD (with l integer) can be solved in polynomial time using dynamic programming. Then by considering multiple alignments of the strips, we can find at least one where the error committed at the joints between strips is small. For clarity we begin by assuming that $d = 2$, the metric is L_∞, and the cluster size is fixed at $D = 1$. So we are trying to cover a set $\mathbf{S}$ of n points in a strip H_l with an optimal number of unit squares having a side parallel to the direction of the strip. For any optimal solution to this problem, there is a solution where the squares are shifted as far right and upwards as possible, as long as the squares remain inside the strip H_l and still contain the points in $\mathbf{S}$ they originally contained.

The strip H_l can be split into two with a vertical segment of height l; we call such a segment a *vertical cut*. For a given candidate solution, the *configuration across a vertical cut* is the set of unit squares in this candidate solution which intersect the cut. The following two lemmas permit us to restrict the number of configurations allowed without losing optimality.

Lemma 9 *Every collection of n points in the strip H_l has an optimal covering in which no more than three squares intersect and no more than $2l-1$ squares intersect the same vertical cut.*

Proof. For each square in the covering, consider the horizontal coordinate of a leftmost point in $\mathbf{S}$ covered by the square. These coordinates, one per square, can be listed in order, from right to left. Consider an optimal covering for which this list of coordinates is lexicographically largest. Suppose that four squares intersect in p. Then we can replace these four squares by the four squares that have p as one of their corners, and still cover all the points in $\mathbf{S}$. Two of these four squares are to the right of p; these two squares can be moved slightly to the right, since the other two squares, to the left of p, already cover the points in $\mathbf{S}$ with the same horizontal coordinate as p that are covered by the two squares to the right. The leftmost point covered by each of the two squares to the right of p now has a horizontal coordinate larger than the horizontal coordinate of p, giving an optimal covering that is lexicographically larger.

Similarly, if $2l$ squares intersect a vertical cut C, whose horizontal coordinate is also denoted by C, then we can replace these $2l$ squares by l squares that cover all the points in H_l whose horizontal coordinate is between $C-1$ and C, and l squares that cover all the points in H_l whose horizontal coordinate is between C and $C+1$. The l squares to the right of C can be moved slightly to the right, since the l squares to the left of C already cover the points in $\mathbf{S}$ whose horizontal coordinate equals C. The leftmost point covered by each of the l squares to the right of C now has a horizontal coordinate larger than C, giving an optimal covering that is lexicographically larger. □

Lemma 10 *The number of possible configurations across a vertical cut satisfying the conditions of Lemma 6.1 is at most* $c(\rho, l) = 4^{l-1}(2\rho)^{2(2l-1)}$*, where* ρ *is the maximum number of points of* $\mathbf{S}$ *in a unit square.*

PROOF. Consider the $l-1$ points $p_1,\ldots,p_{l-1}$ along a vertical cut C, with p_i at distance i from the bottom of the cut. Notice that any square which intersects the vertical cut must contain one of the p_i as well. We can choose how many squares contain each p_i in 4^{l-1} possible ways, since each p_i can only be contained in 0, 1, 2, or 3 squares. Once this choice is made, we can specify each of the squares containing a p_i by giving the lowest and leftmost points in $\mathbf{S}$ it covers. The number of possible choices for each of these two points is at most 2ρ, since each one of them must be in one of the two unit squares having p_i as a corner and located below p_i (to the left of p_i) for the lowest point (resp., for the leftmost point). This gives at most $(2\rho)^2$ possibilities for each square. Since the total number of squares intersecting a vertical cut is at most $2l-1$, we have $(2\rho)^{2(2l-1)}$ possible choices. □

We use this polynomial bound on the number of possible configurations to derive a polynomial time dynamic programming algorithm for our problem.

Theorem 6 *An optimal covering with unit squares for a set of* n *points in the strip* H^l *can be found in* $O(c(\rho, l)ln + n\log n)$ *time.*

PROOF. We sort the points in $\mathbf{S}$ horizontally for each of the l disjoint horizontal strips of width 1 contained in H_l, in $O(n\log n)$ time. We then consider the vertical cuts passing through each one of the n points in order, from left to right. The next cut can always be determined in $O(l)$ time by looking at the next element in each of the l sorted lists. When considering the vertical cut C passing through some point $p \in \mathbf{S}$, we will already have a collection of possible configurations across cut C, and for each such configuration, we will also have the optimal number of squares needed to cover all the points in $\mathbf{S}$ to the left of cut C, including the squares in the configuration itself. If a configuration A in this collection already has a square containing p, then we keep A so that it can be examined by the cut containing the next point in $\mathbf{S}$ to the right of p. Otherwise, we generate valid configurations (if any) by adding to A a square having p as a leftmost point. If point p is between the points p_i and p_{i+1} on cut C at respective distances i and $i+1$ from the bottom of the strip, then any square having p as a leftmost point touches either p_i or p_{i+1}; we only add to A squares touching p_i if p_i is not already contained in three squares in A, and similarly for p_{i+1}. We can easily find all the candidate lowest points for a square whose left border touches p_i (or p_{i+1}) by examining the sorted points in the strip of width 1 below p.

After each configuration for the cut corresponding to a given point $p \in \mathbf{S}$ is generated, we remove from it squares that do not intersect the cut C for the next point in $\mathbf{S}$ to be considered. This elimination can create duplicate configurations, so we keep configurations in an array to detect duplicates. The array is indexed by the number of squares (0,1,2 or 3) containing each of the evenly spaced points $p_1,\ldots,p_{l-1}$ along cut C, and for each square Q out of the total $2l-1$ squares, by two nonnegative numbers smaller than 2ρ that indicate the lowest and leftmost points covered by Q (we associate with each point its position mod ρ in the sorted list for the strip of width 1 it belongs to). The array therefore has $4^{l-1}(2\rho)^{2(2l-1)}$ entries, one for each possible configuration across cut C; the entry for each configuration is the number of squares needed to cover the points up to cut C. Since a configuration has $O(l)$ squares, removing squares and accessing the array takes $O(l)$ time per configuration, and hence, by the preceding lemma, $O(l4^{l-1}(2\rho)^{2(2l-1)})$ time per cut. Since n cuts are considered, the theorem follows. □

For the special case $l = 1$ (the one-dimensional problem) we can do significantly better, obtaining an algorithm that, based on the results of section 3, has the best possible running time:

Theorem 7 *An optimal covering with unit squares for points on a strip of width* 1 *can be found in* $O(n\log k)$ *time, where* k *is the number of squares used.*

PROOF. For strips of width 1, we only need to choose the horizontal placement of the squares, so the problem is one-dimensional and can be viewed as covering points in a horizontal line with intervals of unit length. An optimal solution can be obtained by letting the leftmost interval be the interval whose left endpoint is the leftmost point in **S**, then removing From **S** all the points covered by this interval, and proceeding inductively for the remaining points in **S**.

This algorithm takes $O(n)$ time provided that the points in **S** are sorted From left to right; sorting, however, requires $O(n \log n)$ time. Instead, if r intervals have been selected so far, then the algorithm will maintain a partition of the line into r sectors with $O(n/r)$ points from **S** in each sector, with the sectors sorted from left to right. Whenever a new interval of unit length is selected, the sector with the largest number of points from **S** will be split into two by median-finding, in $O(n/r)$ time; this partitioning rule guarantees that the $O(n/r)$ bound on the number of points in each sector is maintained, because by the time r doubles, the maximum number of points per sector will have decreased by a factor of two. The leftmost point can be found in $O(n/r)$ time by examining the leftmost sector. The total time complexity is thus $\sum_{1 \le r \le k} n/r = O(n \log k)$. □

Remark. The running time for this problem can be reduced to $O(n)$ by grouping points into buckets of size D, and then scanning groups of adjacent occupied buckets. Unfortunately, this requires a potentially infinite array of buckets [Vs87]. This running time does not contradict the lower bounds of section 3, since the floor function and unbounded branching are used in bucketing. The same remark applies below to Theorem 6.4.

These algorithms for finding optimal solutions in a strip can be used to provide approximate solutions in the entire plane. The argument is similar to the one used in Baker's treatment of planar graphs [Ba83].

Theorem 8 *A covering with an approximately optimal number of unit squares for a set of points in the plane can be found in $O(c(\rho, l)l^2 n + n \log n)$ time for an approximation factor of $1 + 1/l$.*

PROOF. We sort all the points $p = (x, y) \in \mathbf{S}$ according to their y value, then group together those points p for which the value of $\lfloor y \rfloor$ is the same. These are points which belong to the same infinite horizontal strip of width 1. We then sort the points in each horizontal strip of width 1 by their x value. Now we generate l different candidate solutions $Sol_0, \ldots, Sol_{l-1}$. To produce solution Sol_i, we group every l consecutive strips of width 1 into a single strip of width l, putting together those points $p = (x, y)$ for which the value of $\lfloor (y + i)/l \rfloor$ is the same. We then obtain a candidate solution Sol_i by putting together the solutions for the disjoint strips of width l. Finally, we choose the candidate solution Sol_i which uses fewer unit squares.

Notice that each square of an optimal solution Sol can only intersect one of the l sets of dividing lines $\mathbf{L}_i = \{(x, y) | y \bmod l = i\}$, for $i = 0, \ldots, l-1$. Choose the value of i for which $\mathbf{L}_i$ intersects fewer squares of the optimal Sol. Then if we create two copies of each square intersected by $\mathbf{L}_i$, we only increase the total number of squares in Sol by a fraction of at most $1/l$. Moreover, this duplication gives us solutions with disjoint collections of boxes for each of the strips of width l determined by $\mathbf{L}_i$, hence a solution no better than Sol_i. Thus Sol_i is within a factor of $1 + 1/l$ of the optimum. Finding l different solutions costs us a factor of l over the running time in the preceding theorem, except for the $O(n \log n)$ time for sorting, which is done only once. □

Here again the $l = 1$ case is considerably faster. We apply the one-dimensional solution twice: first in the y-direction to divide the points into strips and then in the x-direction to process the individual strips. This gives:

Theorem 9 *A covering by unit squares within a factor of 2 of the optimal number of squares can be found in $O(n \log k)$ time.*

These results extend to other L_q metrics and to higher dimensions. We assume as before that the number of dimensions d is a constant. A stip is now the product of an infinite line with a $d-1$-dimensional box whose side-length is a constant times l.

Theorem 10 *A central L_q clustering using a number of clusters within a factor of $1 + 1/l$ of the optimum can be found in $O(c(\rho, l')l' ln + n \log n)$ time, where l' is $O(l_{d-1})$ for a d-dimensional space, and ρ is the maximum density of a hypercube.*

If we are willing to approximate both the number of clusters and the size of clusters simultaneously, then we can remove the potentially expensive ρ term from the running time. First each point is shifted to a nearby joint on a rectilinear grid of size $1/m$. Once duplicates are removed, the density becomes bounded by a constant $\rho = O(m^d)$. Using the above result, we can find a solution for this new problem using at most

$1+1/l$ times the optimal number of clusters, and then increase the cluster size by a factor of $1+1/m$ to cover the original set of points. This gives:

Theorem 11 *A central L_q clustering with a number of clusters within a factor of $1+1/l$ of the optimum and a cluster size within a factor of $1+1/m$ of the optimum can be found in $O(c(m^d, l')l'ln + n\log n)$ time, where l' is $O(l^{d-1})$.*

7. Extensions

The *restricted centers problem* differs from the central clustering problem in that the choice of cluster centers is restricted to points in the set $\mathbf{S}$. We can relax this condition by considering two not necessarily disjoint sets, namely a set $\mathbf{S}_{\text{cust}}$ of customer points to be clustered, and a set $\mathbf{S}_{\text{sup}}$ of supplier points which are the candidate cluster centers: this gives us the *suppliers problem.* If, in either problem, we assign weights to the candidate cluster centers, and generalize the constraint on the number of clusters to a constraint on the sum of the weights of the chosen cluster centers, we obtain the *weighted centers problem* and the *weighted suppliers problem* respectively. (See Hochbaum and Shmoys for a discussion of these problems in graphs [HS86].)

Our NP-hardness proof for the restricted centers problem uses an embedding of an instance of the planar dominating set problem similar to the one in Theorem 2.1 ([GJ79]), but this time we replace edges by paths of length $3i+1$. For the suppliers problem, we again use a reduction from planar vertex cover, the nodes of the graph in Fig. 1 being the candidate suppliers, and the midpoints of the edges the consumers.

Theorem 12 *The restricted centers problem is NP-hard in $d \geq 2$ dimensions for approximation factors of α in the cluster size if $\alpha < \sqrt{3} \approx 1.73$ for L_2 distances, and $\alpha < 2$ for L_1 and L_∞ distances.*

Theorem 13 *The suppliers problem is NP-hard in $d \geq 2$ dimensions for approximation factors of α in the cluster size if $\alpha < \sqrt{7} \approx 2.65$ for L_2 distances, and $\alpha < 3$ for L_1 and L_∞ distances.*

The algorithm of sections 4 and 5 also solves the restricted centers problem with an approximation factor of 2. We can solve the more general of these problems, the weighted suppliers problem, with an approximation factor of 3. The idea is to run, in the first phase of the algorithm, two box decompositions, one for customer points and one for supplier points. The customer boxes are maintained as before; however, when we maintain the supplier boxes, we discard portions that are far from customer boxes, in order to avoid creating more than $O(k)$ supplier boxes. It is also necessary to keep track of the least weight in each supplier box: here the technique used in section 5 for ordering the coordinates of points is used to keep the weights ordered in each box. The first phase achieves a crude approximation factor of 9, while the second phase produces a more accurate estimate by maintaining the exact distances between customers and nearby suppliers, while keeping them only partially sorted to achieve efficiency.

Theorem 14 *The weighted suppliers problem can be solved with an approximation factor of 3 for the cluster size in $O(n\log k)$ time, under any given L_q distance metric in d-dimensional space.*

If we run the farthest point algorithm among customer points, keeping track for each customer $p_i \in \mathbf{T}$ of the supplier q_i with least weight within distance $D/2$ of p_i, until $\Sigma w(q_i) > W$, we obtain at the end of the algorithm a lower bound of D for the cluster size, and also an upper bound of $3D$ since the last time $\Sigma w(q_i) \leq W$ all customer points are within distance $3D/2$ of some q_i.

We obtain again a more efficient algorithm by grouping points into boxes, but we maintain two separate collections of boxes, $\mathbf{B}_{cust}$ and $\mathbf{B}_{sup}$, for customer and supplier points, all boxes of length at most D'/d. We also have a graph $G^{(1)}_{D'}$ to compute a maximal independent set $\mathbf{T}'$ between customer boxes, and a bipartite graph $G^{(2)}_{D'/2}$ between customer and supplier boxes to choose suppliers with small weights near each box in $\mathbf{T}'$. This gives a lower bound of D', and also an upper bound of $9D'$, since all customer boxes are within distance D' of a box in $\mathbf{T}'$ which in turn is within distance $D'/2$ of a supplier box containing chosen supplier of small weight, and the diameter of boxes is at most D'. Two other provisions are needed for efficiency. First, we must keep supplier weights for each supplier box partially sorted, just as we did for coordinates in the implementation of the box decomposition, so as to be able to determine the least weight in a supplier box efficiently. Secondly, when shrinking a supplier box, we leave out areas of the box which could not contain suppliers within distance $D'/2$ of a customer box. This makes the number of supplier boxes remain $O(k)$, because the number of consumer boxes is $O(k)$, so that the running time remains within $O(n \log k)$.

A second phase reduces the approximation factor from 9 to 3. We use graphs $G^{(1)}_{3D'}$ and $G^{(2)}_{3D'/2}$ constructed during the first phase. The graph $G^{(1)}_{3D'}$ allows us to run a farthest point algorithm among customer points while only updating points in neighbor boxes. If for each customer $p_i \in \mathbf{T}''$ we could sort according to their distance to p_i the n_i suppliers at boxes adjacent to the box containing $p\Delta i$ in $G^{(2)}_{3D'/2}$, it would be easy to keep track of the supplier with least weight within distance $D''/2$ of $p\Delta i$ for $D' \leq D'' < 3D'$. Sorting, however, takes $O(n \log n)$ time, so we must again keep the $n\Delta i$ suppliers only partially sorted into groups of n_i/k suppliers each in order to update supplier weights for the customers $p_i \in \mathbf{T}''$ in $O(n \log k)$ time. When the farthest point set $\mathbf{T}''$ reaches its final size, a binary search on the $\Sigma n_i = O(n)$ distances between the $p_i \in \mathbf{T}''$ and their n_i candidate suppliers gives us in $O(n)$ time a lower bound of D'' and a solution which achieves $3D''$.

Once we assign two (or more) weights to each candidate cluster center and impose constraints on the sum of the weights of the chosen cluster centers for each of the two sets of weights, we can no longer obtain good approximation bounds:

Theorem 15 *The two-weighted centers problem remains NP-hard in one-dimensional space for an arbitrarily large approximation factor α.*

8. Conclusion

We have shown that it is possible to approximate the solution of geometric clustering problems in two directions. The optimal number of clusters can be approximated in polynomial time, with the cost of computing the approximation becoming expensive as the approximation factor nears 1. By contrast, the cost of approximating the optimal cluster size shows a much more dramatic transition: the problem is NP-hard unless the approximation factor is near 2, and then at 2 the approximation can be computed in $O(n \log k)$ time, matching an algebraic decision tree lower bound. The transition at 2 is tight for the L_1 and L_∞ metrics, but still contains a small unsettled gap for the L_2 metric. The techniques used to obtain the optimal running time for approximate cluster size can be extended to a variety of weighted supplier–customer problems.

Acknowledgments

We would like to thank Frances Yao, Mike Paterson, and Marshall Bern for helpful suggestions during this work.

References

[Ba83] Brenda S. Baker, "Approximation algorithms for NP-complete problems on planar graphs," Proc. 24th IEEE FOCS (1983), pp. 265–273.

[Be83] Michael Ben-Or, "Lower bounds for algebraic computation trees," Proc. 15th ACM STOC (1983), pp. 80–86.

[Br77] P. Brucker, "On the complexity of clustering problems," in Optimization and Operations Research (R. Henn, B. Korte, W. Oletti eds.), Springer-Verlag, Berlin, Heidelberg, New York (1977), pp. 45–54.

[Eq87] William Equitz, "Fast algorithms for vector quantization picture coding," Proc. International Conference on Acoustics, Speech, and Signal Processing (1987), pp. 725–728.

[FPT81] R.J. Fowler, M.S. Paterson and S.L. Tanimoto, "Optimal packing and covering in the plane are NP-complete," Inf. Proc. Letters, Vol. 12, No. 3 (1981), pp. 133–137.

[GJ77] M.R. Garey and D.S. Johnson, "The rectilinear Steiner tree problem is NP-complete," Siam J. Appl. Math. 32 (1977), pp. 826–834.

[GJ79] M.R. Garey and D.S. Johnson, "Computers and intractability: a guide to the theory of NP-completeness," W.H. Freeman, San Francisco (1979).

[Go85] Teófilo F. González, "Clustering to minimize the maximum intercluster distance," Theoretical Computer Science 38, North-Holland (1985), pp. 293–306.

[Gr84] Robert M. Gray, "Vector quantization," IEEE ASSP Magazine, April (1984), pp. 4-29.

[HS85] D.S. Hochbaum and D.B. Shmoys, "A best possible heuristic for the k-center problem," Mathematics of Operations Research, Vol. 10, No. 2 (1985), pp. 180–184.

[HS86] D.S. Hochbaum and D.B. Shmoys, "A unified approach to approximation algorithms for bottleneck problems," JACM, Vol. 33, No. 3 (1986), pp. 533–550.

[HN79] W.L. Hsu and G.L. Nemhauser, "Easy and hard bottleneck location problems," Discr. Appl. Math. 1 (1979), pp. 209–215.

[MS84] N. Megiddo and K.J. Supowit, "On the complexity of some common geometric location problems," Siam J. Comput., Vol. 13, No. 1 (1984), pp. 182–196.

[Me88] Stuart G. Mentzer, "Lower bounds on metric k-center problems," Manuscript, 1988.

[PS85] F.P. Preparata and M.I. Shamos, "Computational geometry: an introduction," Springer-Verlag, New York, Berlin, Heidelberg, Tokyo (1985).

[Va86] Pravin M. Vaidya, "An optimal algorithm for the all-nearest-neighbors problem," Proc. 27th IEEE FOCS (1986), pp. 117–122.

[Vs87] Stephen Vavasis, personal communication.