Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik I

Roland Kindermann

Kostenoptimale Überdeckung von Punktmengen durch Kreise

21. März 2008

Seminararbeit im WS 2006/2007

Zusammenfassung

In dieser Ausarbeitung wird folgendes Problem betrachtet: Gegeben ist eine Menge von Punkten $Y = \{p_1, \ldots, p_n\} \subset \mathbb{R}^2$. Gesucht wird eine Menge $\mathcal{D} = \{D_1, \ldots, D_m\}$ von Kreisen mit Radien r_i und Zentren t_i, sodass jeder Punkt aus Y von mindestens einem Kreis überdeckt wird. Dabei soll die Kostenfunktion $\sum_{i=1}^{m} f(r_i)$ mit $f(r) = r^\alpha$ minimiert werden. So wird beispielsweise für $\alpha = 1$ die Summe der Radien und für $\alpha = 2$ die überdeckte Fläche minimiert. Dabei können verschiedene Einschränkungen für die Wahl der Mittelpunkte der Kreise gelten. Je nach Problemstellung wird dabei die euklidische Metrik oder eine andere L_p-Metrik verwendet.

Folgende Resultat werden vorgestellt:

- Für eindimensionale Variante mit einer diskreten Menge gültiger Kreiszentren werden Algorithmen zur exakten Lösung des Problems angegeben.
- Es wird bewiesen, dass das zweidimensionale Problem bei einer diskreten Menge gültiger Zentren NP-hart ist.
- Es werden exakte und approximative Algorithmen für auf eine Gerade beschränkte Kreiszentren angegeben.
- Es wird gezeigt, dass eine Variante des Problems, bei der die Zentren auf einer frei wählbaren, achsenparallelen Gerade liegen müssen, unberechenbar durch Wurzeln ist. Anschließend wird ein Approximationsalgorithmus für das Problem angegeben.
- Approximationsalgorithmen für auf einer frei wählbaren Geraden liegende Kreiszentren werden vorgestellt.

Inhaltsverzeichnis

1 Einführung

Die folgende Ausarbeitung basiert auf [2] und beschäftigt sich mit dem folgenden Problem:
Gegeben sei $Y = \{p_1, \ldots, p_n\} \subset \mathbb{R}^2$, eine Menge von Punkten in der Ebene und eine Kostenfunktion $f(r) = r^\alpha$. Ein Kreis D mit Zentrum in t und Radius r überdecke einen Punkt p_i (bei gegebener L_p-Metrik $d(.,.)$) genau dann, wenn $d(t, p_i) \leq r$. Gesucht wird eine Menge von Kreisen $\mathcal{D} = \{(t_1, r_1), \ldots, (t_m, r_m)\}$, mit Zentren t_i und Radien r_i, sodass jeder Punkt aus Y durch mindestens einen Kreis von $\mathcal{D}$ überdeckt wird, und die Gesamtkosten $\sum_{i=1}^{m} f(r_i)$ minimiert werden. Offensichtlich ist die Lösung des Problems in dieser Form trivial: Werden n Kreise mit Radius 0 auf den Punkten aus Y platziert, so sind die Gesamtkosten 0. Aus diesem Grund gibt es in verschiedenen Szenarien verschiedene Einschränkungen für die gültigen Positionen der Zentren der Kreise.

Das Problem entspricht einem Szenario, in dem Sender so platziert werden müssen, dass eine Menge von Empfängern in Reichweite der Sender liegen. Die Radien entsprechen in diesem Szenario den Sendeleistungen der Sender. Angelehnt an diese Anwendung werden die Punkte in Y auch Clients und die Kreise auch Server genannt.

Soll die Kostenfunktion die benötigte Energie in Abhängigkeit der Reichweite modellieren, so werden üblicherweise superlineare Funktionen ($\alpha > 1$) verwendet. Verwendet man $\alpha = 2$, so entspricht dies der überdeckten Fläche. Lineare Kosten entsprechen der Summe der Radien und werden beispielsweise in [20] verwendet. Lineare Kosten werden nicht nur verwendet, um das Problem zu vereinfachen und Erkenntnisse über die Struktur des Problems zu gewinnen, sondern auch um Situationen zu modellieren, in denen tatsächlich lineare Kosten auftreten [11, 21]. Beispiele für solche Situationen sind:

- Übertragungen werden mittels eines schmalen Strahls, der entweder gedreht, oder bei Bedarf ausgerichtet wird, durchgeführt.
- Ein Roboter soll seine Umgebung unter Zuhilfenahme eines rotierenden Laserstrahls scannen [15, 16]. Möchte man eine feste räumliche Auflösung erhalten, so hängt der Zeitaufwand des Scannens linear vom Radius ab.

Das Problem ist ein Clustering-Problem und wurde von Bilò et al. [8] „min-size k-clustering problem“ genannt. Da viele Clustering-Probleme NP-hart sind, wird häufig versucht, Approximationsalgorithmen oder Polynomial Time Aproximation Schemes ($PTAS$) zu finden.

Ein Approximationsalgorithmus mit Approximationsgüte c ist ein Algorithmus, der eine Lösung des Problems findet, die höchstens die c-fachen Kosten der optimalen Lösung OPT hat, wobei c eine von der Größe der

Eingabe unabhängige Konstante ist. Seien OPT die Kosten der optimalen Lösung. Ein PTAS ist ein Algorithmus, der zu jedem $\epsilon > 0$ eine Lösung mit Kosten $\leq (1+\epsilon)OPT$ findet, und dessen Laufzeit polynomiell von der Größe der Eingabe abhängt. Ein PTAS heißt Fully Polynomial Time Aproximation Scheme ($FPTAS$), wenn die Laufzeit außerdem maximal polynomiell von $\frac{1}{\epsilon}$ abhängt. Alle in dieser Ausarbeitung vorgestellten PTAS sind FPTAS.

In [2] wird auch eine „minimum cost covering tour“ ($MCCT$) genannte Abwandlung des Problems vorgestellt, bei der die Kreiszentren auf den Ecken eines Polygons liegen und die Gesamtkosten einer Linearkombination aus der Summe der Kosten der Kreise und der Länge des Randes des Polygons sind. In dieser Ausarbeitung wird dieses Problem allerdings nicht betrachtet.

1.1 Einschränkung der Positionen

Wie bereits erwähnt, ist die Lösung des Problems trivial, wenn keine Beschränkungen für die Positionen der Server existieren. Aus diesem Grund kann beispielsweise die Anzahl der Server auf $k < n$ beschränkt werden. In dieser Ausarbeitung werden jedoch nur Varianten betrachtet, bei denen die gültigen Serverpositionen(die Zentren der Kreise) eingeschränkt sind. Dabei gibt es zwei grundlegende Arten der Einschränkung:

1. Die Serverpositionen müssen Elemente einer diskreten Menge $X = \{t_1, \ldots, t_m\}$ möglicher Positionen sein.

2. Die Serverpositionen müssen auf einer Gerade liegen. Je nach Variante ist diese Gerade vorgegeben oder darf achsenparallel oder vollkommen frei gewählt werden.

1.2 Vorgestellte Resultate

Es werden mehrere Resultaten - teils Verbesserungen bisheriger Resultate und teils neue Ergebnisse - vorgestellt.

Für die auf eine diskrete Menge beschränkte Menge werden die Resultate von [8, 20] verbessert. Für das diskrete, eindimensionale Problem ($Y \subseteq \mathbb{R}$) wird „Closest-Center-with-Growth“ (CCG), eine 3-Approximation, also ein Approximationsalgorithmus mit Approximationsgüte 3, mit linearer Zeitkomplexität vorgestellt. Desweiteren wird der „Greedy Growth“ (GG) Algorithmus, der eine 2-Approximation in beinahe linearer Zeit berechnet, vorgestellt.

Vor Erscheinen des Artikels [2] war bereits bekannt, dass die 2D-Varianten des Problems für $\alpha > 2$ NP-hart ist, und dass ein FPTAS existiert [8]. Hier wird gezeigt, dass das zweidimensionale Problem mit einer diskreten Menge potentielle Serverpositionen bereits für $\alpha > 1$ NP-hart ist.

Anschließend werden Varianten des Problems, bei denen weniger strenge Restriktionen für die Serverpositionen gelten, betrachtet. Es werden Dynamisches-Programmieren-Algorithmen präsentiert, die das Problem bei auf eine Gerade beschränkten Serverpositionen für beliebige L_p-Metriken bei linearen Kosten in $O(n^2 \log n)$ und für beliebige superlineare, monoton steigende Kostenfunktionen in $O(n^4 \log n)$ Zeit lösen. Desweiteren wird der „Squar- Greedy"-Algorithmus (SG), der in $O(n \log n)$ eine 3-Approximation für lineare oder superlineare Kosten unter der Verwendung der L_∞-Metrik berechnet, vorgestellt. Anschließend wird der „Square Greedy with Growth" Algorithmus, der bei linearen Kosten unter Verwendung der L_∞-Metrik in $O(n \log n)$ Zeit eine 2-Approximation berechnet, präsentiert. Danach wird gezeigt, wie die Resultate sich auf andere L_p-Metriken verallgemeinern lassen.

Ein Szenario, in dem die Beschränkung der Serverpositionen auf eine Gerade vorkommt, ist beispielsweise eine Situation, in der die Server entlang einer Autobahn platziert werden sollen. Soll außerdem der optimale Verlauf der Autobahn bestimmt werden, so enthält die Problemstellung das Problem der optimalen Wahl der Geraden. Ein anderes solches Szenario ist das Platzieren von Geräten entlang eines Laser- oder Mikrowellen-Strahls, aus dem sie ihre Energie beziehen.

Es wird gezeigt, dass, wenn die Server entlang einer frei wählbaren horizontalen Geraden platziert werden müssen, die exakte, optimale Lösung für $\alpha = 1$ unberechenbar durch Wurzeln ist. Dies geschieht unter Verwendung einer Technik, die ähnlich dem Beweis der Unlösbarkeit des Fermat-Weber-Problems [6, 7] funktioniert. Außerdem wird ein FPTAS, dass für $\alpha = 1$ $O(\frac{n^3}{\epsilon} \log n)$ und für $\alpha > 1$ $O(\frac{n^5}{\epsilon} \log n)$ Zeit benötigt, angegeben.

Für das Problem einer frei wählbaren Geraden und linearen Kosten werden $O(1)$-Approximationen und ein FPTAS mit Zeitkomplexität$O(\frac{n^5}{\epsilon^2} \log n)$ vorgestellt.

1.3 Verwandte Arbeiten

Es gibt eine ganze Reihe von Clustering-Problemen, beispielsweise das *k-center* Problem, bei dem das Maximum der Radien minimiert wird, das *k-median* Problem, bei dem die Summe der Abstände der Zentren der Kreise von den überdeckten Clients minimiert wird und das *k-clustering* Problem, bei dem über alle Cluster das Maximum der Summen der paarweisen Entfernungen innerhalb des Clusters minimiert wird. Veröffentlichungen, die sich mit diesen Problemen beschäftigen, sind [1, 5, 18]. Für das Clustering-Problem, bei dem bis zu k Cluster platziert werden dürfen und die Summe ihrer Radien minimiert werden soll, wird eine $O(1)$ Approximation in [10] angegeben.

[20] beschäftigt sich mit der eindimensionalen, diskreten Version des in dieser Ausarbeitung betrachteten Problems. Unter den Resultaten finden

sich ein $O((n+m)^3)$ Algorithmus zur exakten Lösung des Problems und eine 4-Approximation mit linearer Zeitkomplexität. Dieselbe Veröffentlichung enthält auch einen PTAS für das zweidimensionale Problem.

Bilò et al. [8] zeigen, dass das zweidimensionale Problem mit Kostenfunktion $f(r) = r^\alpha$ und $\alpha > 2$ NP-hart ist, wenn entweder die Serverpositionen auf eine diskrete Menge beschränkt sind, oder nur $k < n$ Kreise mit *Client*positionen als Mittelpunkt platziert werden dürfen. Desweiteren werden ein PTAS für $\alpha = 1$ und eines für ein allgemeineres Problem, bei dem die Kosten superlinear sind, zusätzliche Kosten für jeden Server addiert werden und die Anzahl der Server beschränkt ist, vorgestellt.

Es existieren auch viele Probleme, bei denen die Clients durch Kreise eines festen Radius überdeckt werden müssen. Hochbaum und Maas [19] geben ein PTAS für eine Überdeckung mit einer minimalen Anzahl von Kreisen mit festem Radius und beliebigen Zentren an. Dabei führen sie eine „grid-shifting“ genannte Technik ein, die in [8, 14, 20] verwendet und verbessert wird.

Für auf eine diskrete Menge X beschränkte Serverpositionen betrachtet Gonzales [17] ein Problem, bei dem die Anzahl der überdeckten Clients maximiert und die Anzahl der Server minimiert werden muss. Er gibt ein PTAS für die Lösung solcher Probleme bei denen zusätzlich ein Mindestabstand zwischen den Servern liegen muss an.

In [9] wird eine c-Approximation mit polynomieller Laufzeit für ein Problem vorgestellt, bei dem aus einer Menge mögliche Server ausgewählt werden müssen, sodass diese eine Menge von Clients überdecken und die Gesamtkosten minimiert werden.

Die dem MCCT ähnlichsten Probleme sind das in [3] präsentierte „lawnmower“-Problem und das in [4, 13] beschriebene Travelling-Salesment-Problem mit Nachbarschaften. Beide sind NP-hart und für beide existieren verschiedene Approximationsalgorithmen.

2 Szenario 1: Auf eine diskrete, eindimensionale Menge beschränkte Serverstandorte und lineare Kosten

Zunächst wird eine eindimensional, diskrete Variante des Problems betrachtet. Die Eingabe des eindimensionalen, diskreten Problems mit lineare Kosten besteht aus einer diskreten Menge $X = \{t_1, \ldots, t_m\}$ von m möglichen Serverpositionen und einer Menge $Y = \{p_1, \ldots, p_n\}$ von n zu überdeckende Clientpositionen. Die Server- und Client-Positionen liegen entlang einer Geraden. Gesucht ist eine minimale Überdeckung der Clients durch Kreise, deren Mittelpunkte in X liegen. Dabei wird von einer linearen Kostenfunktion ($\alpha = 1$) ausgegangen. Es wird angenommen, dass X und Y in der

gleichen Reihenfolge sortiert sind. Sollte dies nicht der Fall sein, kann die Sortierung in $O((n+m)\log(n+m))$ hergestellt werden.

Bisherige Ergebnisse von Lev-Tov und Peleg enthalten einen Algorithmus, der in $O((n+m)^3)$ die exakte Lösung liefert [20]. Desweiteren zeigten Bilò et all, dass das Problem für beliebiges α in polynomieller Zeit gelöst werden kann [8]. Die Zeitkomplexität der dabei verwendeten Algorithmen ist zwar polynomiell, aber dennoch relativ hoch. Es gibt einen weiteren Algorithmus von Lev-Tov und Peleg, der in linearer Zeit eine 4-Approximation berechnet [20].

Im folgenden werden ein Algorithmus, der in linearer Zeit eine 3-Approximation berechnet, und einer, der in $O(m + n\log m)$ Zeit eine 2-Approximation berechnet, vorgestellt.

2.1 Der Closest-Center-with-Growth-Algorithmus (*CCG*)

Die Idee des Closest-Center-with-Growth-Algorithmus ist, die Clients von links nach rechts zu verarbeiten und dabei zu entscheiden, ob es sinnvoller ist, einen bereits existierenden, links vom aktuellen Client liegenden Kreis zu vergrößern, oder einen neuen Kreis rechts vom Client einzufügen.

Sei p_i ein Client und seien alle Clients links von p_i bereits durch den CCG-Algorithmus überdeckt worden. Dann gibt es drei Möglichkeiten:

- p_i wurde bereits überdeckt und muss nicht weiter betrachtet werden.
- CCG fängt p_i mit einem Kreis von links ein, vergrößert ihn also so weit, dass er p_i überdeckt. Dabei wird derjenige Kreis vergrößert, bei dem die durch das Einfangen verursachten Zusatzkosten am geringsten sind. Da alle bereits platzierten Kreise links von p_i liegen, ist dies offensichtlich der Kreis, dessen Rand am weitesten rechts liegt. Da alle Clients links von p_i bereits eingefangen wurden, werden bei dieser Aktion keine neuen Clients als Seiteneffekt überdeckt.
- CCG fängt p_i durch einen neuen Kreis, dessen Zentrum t_k rechts von p_i liegt, ein. Dabei wird der Kreis die Serverposition t_k gelegt, bei der das Platzieren eines Kreises, der p_i überdeckt, die geringsten Kosten verursacht. Folglich ist t_k immer die am weitesten links auf der rechten Seite von p_i liegende Serverposition. Bei der Aktion werden alle Clients in $(p_i, t_k + 2(t_k - p_i)]$ als Seiteneffekt überdeckt.

Der Algorithmus verfährt also von links nach rechts und prüft bei jedem Client p_i, ob er bereits überdeckt wurde. Falls dies nicht der Fall ist, vergrößert er den am weitesten rechts endenen Kreis auf der linken Seite von p_i oder legt einen neuen Kreis um den rechts neben p_i liegenden Server, falls dies billiger ist. Dabei wird implizit angenommen, dass an allen Serverpositionen links von p_i Kreise (gegebenenfalls mit Radius 0) liegen.

Der Algorithmus itertiert über eine sortierte Liste aller Server und Clients. Die Serverliste wird dabei benötigt, um den rechten Nachbarn eines Clients zu bestimmen. Liegt der auf p_i folgende Client p_{i+1} rechts des rechten Nachbarn t_k von p_i, so muss in der Serverliste beginnend von t_k solange gesucht werden, bis ein rechter Nachbar von p_{i+1} gefunden wird. Diese Suchvorgänge benötigen insgesamt $O(m)$ Zeit. Des Weiteren wird in einer Variable gespeichert, welches der am weitesten rechts endende Client ist. Sie muss aktualisiert werden,

- wenn ein Client von rechts überdeckt wird. Der neu platzierte Kreis ist dann der am weitesten rechts liegende.
- wenn auf der Suche nach dem rechten Nachbarn des nächsten Clients der Zeiger in der Serverpositionenliste auf die nächste Serverposition verschoben wird. Dies entspricht dem Platzieren eines Kreises mit Radius 0.

Insgesamt sind also maximal $O(n+m)$ Aktualisierungen nötig, die jeweils in $O(1)$ durchgeführt werden können. CCG benötigt also insgesamt $O(n+m)$ Zeit sofern die Client- und Serverpositionen bereits sortiert vorliegen.

Lemma 1 *Der Closest Center with Growth Algorithmus berechnet eine 3-Approximation der optimalen Lösung in $O(n+m)$ Zeit.*

Dass die Approximationsgüte nicht besser als 3 sein kann, verdeutlicht das folgende, auch in Abbildung 1 dargestellte Beispiel: Seien Serverpositionen bei -2, 0 und 2 gegeben. Die Clients liegen an den Positionen $-1-\epsilon$, $1+\epsilon$ sowie gleichmäßig verteilt mit Abständen $\frac{\epsilon}{2}$ im Intervall $[-1-\epsilon, -\epsilon]$. Es gelte $\frac{1}{2} > \epsilon > 0$. GGC wird nun zunächst den Client bei $-1-\epsilon$ durch einen Kreis mit Radius $1-\epsilon$ und Zentrum bei -2 überdecken. Anschließend wird dieser Kreis vergrößert, so dass das Intervall $[-1-\epsilon, -\epsilon]$ überdeckt wird, da das Einfangen durch den Kreis von links eines Client auf diesem Intervall nur Kosten von $\frac{\epsilon}{2}$ verursacht, während ein Einfangen von rechts mindestens ϵ kosten würde. Zuletzt wird noch der Client bei $1+\epsilon$ durch einen Kreis mit Zentrum bei 2 eingefangen. Die Kosten dieser Lösung betragen dann $2-\epsilon+1-\epsilon = 3-2\epsilon$. Optimal wäre es, alle Clients durch einen Kreis mit Radius $1+\epsilon$ und Mittelpunkt bei 0 zu überdecken. Die Approximationsgüte beträgt also mindestens $\frac{3-2\epsilon}{1+\epsilon}$. Dieser Wert kommt für beliebig kleines ϵ beliebig nah an 3.

Der Beweis der oberen Schranke der Approximattionsgüte ist ähnlich dem von Lemma 2. Im Anschluss an den Beweis von Lemma 2 wird gezeigt, wie er auf Lemma 1 übertragen werden kann.

2.2 Der-Greedy-Growth-Algorithmus (GG)

Der Greedy Growth Algorithmus beginnt mit je einem Kreis mit Radius 0 an jeder Serverposition. Bei jeder Iteration wird unter allen Clients die noch

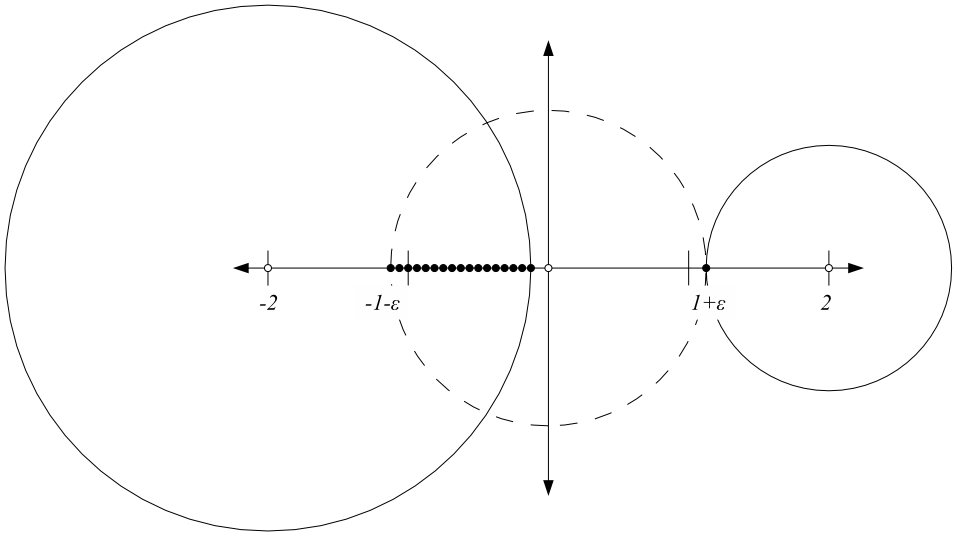


Abbildung 1: Eine Eingabe, bei der der CCG-Algorithmus beinahe das Dreifache der optimalen Kosten berechnet. Der gestrichelte Kreis ist die optimale; der durchgezeichnete die von CCG berechnete Lösung. Die weißen Kreise entsprechen potentiellen Serverpositionen, während die schwarzen Clients repräsentieren.

nicht überdeckt wurden, derjenige gesucht, der mit den geringsten Kosten eingefangen werden kann, also den geringsten Abstand zu einem nächsten Kreis hat.

Für eine effiziente Implementation ist es insbesondere wichtig, schnell zu bestimmen, welcher Client als nächstes eingefangen werden muss. Hierfür wird eine Priority-Queue verwendet. Eine wichtige Beobachtung ist, dass zu jedem Zeitpunkt jeder Kreis K maximal zwei Clients als nächstes einfangen kann – seine beiden Nachbarn. Deshalb müssen auch nur Clients, die direkt mit einem Kreis benachbart sind, in der Priority-Queue gespeichert werden. Bei der Initialisierung müssen pro Serverposition also nur betrachtet werden, ob direkt links oder rechts davon Clientpositionen liegen. Da die Client- und Serverpositionen(gemeinsam in einer großen Liste) sortiert sind, ist dies in konstanter Zeit möglich. Also können alle relevanten Clients zu Beginn in $O(m)$ gefunden werden. Da eine Priority-Queue mit maximal $2m$ Elementen in $O(m)$ initialisiert werden kann, wird also für das Initialisieren insgesamt $O(m)$ Zeit benötigt.

Als nächstes wird die maximale Anzahl der Elemente, die auf einmal in der Priority-Queue liegen können, betrachtet: Da die Priority-Queue genutzt wird, um den Client zu finden, der den geringsten Abstand zum nächsten Kreis hat, müssen wir auch nur Clients, die dafür in Frage kommen einfügen. Alle Punkte, die nicht direkt mit einem Kreis benachbart sind, können also weggelassen werden. Folglich müssen sich zu jedem Zeitpunkt nur bis zu $2m$

Clients in der Priority-Queue befinden. Es sind also zu jedem Zeitpunkt nur $O(m)$ Elemente in der Priority-Queue und die Aktionen „Einfügen in die Priority-Queue“, „Löschen aus der Priority-Queue“ und „Verringern eines Wertes aus der Priority-Queue“ haben die Zeit-Komplexität $O(\log m)$.

Als nächstes betrachten wir die Vergrößerung eines Kreises. Dabei müssen die Werte in Priority-Queue aktualisiert werden. Um die dafür benötigte Zeit abzuschätzen, betrachten wir eine einzelne Vergrößerung. Ohne Beschränkung der Allgemeinheit liege der dabei eingefangene Client p_i links des Zentrums des vergrößerten Kreises K. Zunächst einmal muss p_i aus der Priority-Queue entfernt werden. Liegt links von p_i ein weiterer Client, so können wir ihn aufgrund der Sortierung der Server und Clients in $O(1)$ finden. Wird ein solcher Client gefunden, so muss er in die Priority-Queue eingefügt werden. Sei p_j der nächste Nachbar rechts von K. Dann verringert das Vergrößern von K den Abstand zu p_j und der Schlüssel von p_j in der Priority-Queue muss verringert werden. p_j kann nicht als Seiteneffekt des Vergrößern eingefangen werden, da der Abstand zwischen p_j und K vor dem Vergrößern des Kreises größer war, als der zwischen p_i und K. Eine Ausnahme besteht, wenn K genau gleich weit entfernt von p_i und p_j war. In diesem Fall können wir den Schlüssel von p_j einfach auf 0 setzen. Das einfangen eines Clients benötigt also maximal 3 Priority-Queue-Aktionen, die je $O(\log m)$ Zeit benötigen. Also kostet das Einfangen *aller* Clients insgesamt $O(n \log m)$ Zeit. Werden die Kosten der Initialisierung addiert, ergibt sich eine Gesamtkomplexität von $O(m + n \log m)$.

Lemma 2 *Für $\alpha = 1$ berechnet der Greedy Growth-Algorithmus eine 2-Approximation der Optimalen Lösung OPT in $O(m + n \log m)$ Zeit.*

Beweis. Die Zeitkomplexität wurde bereits gezeigt. Es bleibt also die Approximationsgüte zu beweisen. Die Idee des Beweises ist, für einen Kreis D aus der optimalen Lösung alle enthaltenen Clients zu betrachten. Es wird gezeigt, dass der GG Algorithmus für ihre Überdeckung maximal $2 \cdot Radius(D)$ Kosten benötigt.

Zunächst jedoch seien die Intervalle J_i definiert wie folgt; sei p_i ein Client, der von GG durch den Server t_j eingefangen wird. Der Radius des Kreises um t_j vor dem Einfangen sei r_j. Falls $p_i > t_j$ sei $J_i = (t_j + r_j, p_i]$, andernfalls sei $J_i = [r_i, t_j - r_j)$. Anschaulich ist J_i also die Strecke, die der Kreis p_i „entgegenkommen“ musste. Zwei wichtige Beobachtungen sind, dass die J_i disjunkt sind, also $J_i \cap J_k = \emptyset$ für $i \neq k$, und dass die Summe der Längen der J_i die Summe der Radien ist. Dies liegt daran, dass für jede Vergrößerung eines Kreises ein J_i mit entsprechender Länge erzeugt wird.

Sei D ein Kreis aus der optimalen Überdeckung dessen Zentrum in t_D liegt. Sein Radius sei r_D. Betrachtet werden alle J_i, deren Clients p_i innerhalb von D liegen. Da die J_i disjunkt sind, kann auf jeder Seite des Kreises maximal ein J_i aus dem Kreis herausragen. Soweit existent sei das links

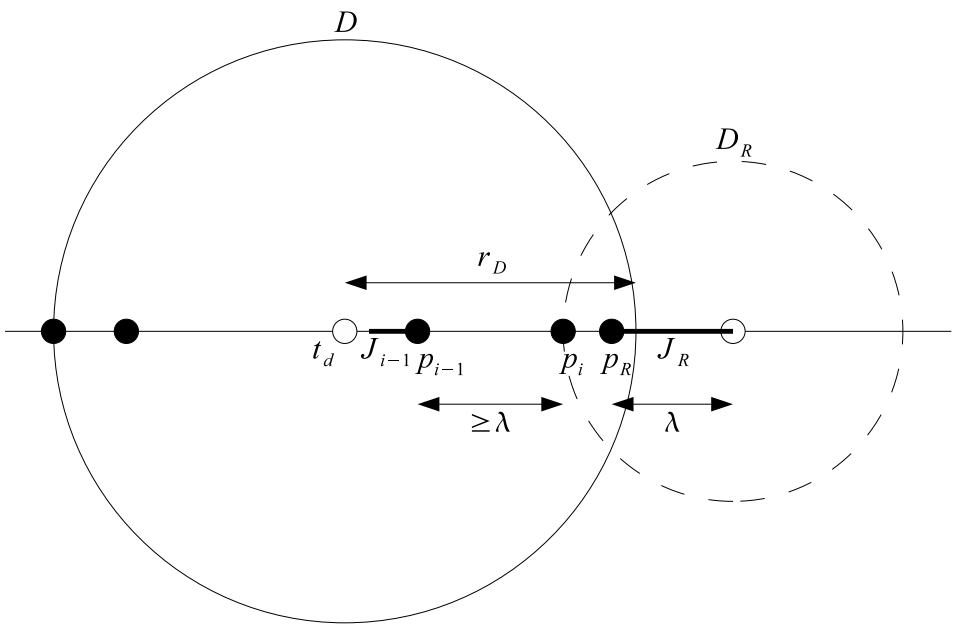


Abbildung 2: Existiert J_R, so existiert zwangsläufig innerhalb der rechten Hälfte von D ein Intervall mindestens der gleichen Länge, dass keine J_k enthält oder schneidet.

herausragende J_i mit J_L bezeichnet und das rechts herausragende mit J_R. Angenommen, J_R existiert. Dann kann J_R nicht um mehr als r_D aus D herausragen, da der zugehörige Client p_R sonst von D eingefangen worden wäre.

Es sei $\lambda = Länge(J_R)$. Im Folgenden wird angenommen, dass J_R existiert. Es wird gezeigt, dass dann innerhalb der rechten Hälfte von D ein Intervall mindestens der Länge λ liegt, dass keine J_i enthält. Aufgrund der Symmetrie des Algorithmus lässt die Aussage sich auch auf J_L übertragen. Da die J_i disjunkt sind folgt dann, dass zu den Clients, die von D überdeckt werden, J_i gehören, deren Länge insgesamt maximal $Durchmesser(D) = 2r_D$ beträgt. Dies gilt auch wenn, J_R nicht existiert da dann trivialerweise die Gesamtlänge der rechts von t_D liegenden J_i, die zu von D überdeckten Clients gehören, maximal r_D beträgt. Es folgt, dass die von D überdeckten Clients in der von GG berechneten Lösung maximal $2 \cdot r_D$ Kosten verursachen, und dass die von GG berechnete Lösung maximal die doppelten Kosten der optimalen Lösung hat.

Es bleibt also folgendes zu zeigen: Angenommen J_R existiert, dann liegt innerhalb von D, rechts von t_D ein Intervall mit Länge $\geq \lambda$, dass keine J_i überschneidet. Da wir von einer Situation ausgehen, in der J_R existiert, vergrößert GG einen Kreis D_R rechts von p_R mindestens so lange, bis p_R eingefangen wurde. Aufgrund anderer Clients kann der Kreis auch danach

noch weiter wachsen. Auf jeden Fall gibt es einen Client p_i in D der zuletzt von D_R *aktiv* eingefangen wird. Mitunter aber nicht zwangsläufig ist dieser Client p_R. p_i ist nicht zwangsläufig der am weitesten links liegende Client in D_R, da nach rechts gerichtetes Wachstum dazu führen kann, dass auf der linken Seite von D_R Clients, die zuvor von einem anderen Kreis eingefangen wurden, auch von D_R überdeckt werden.

p_i muss rechts von t_D liegen, da es sonst weniger Kosten verursacht hätte, p_i mit t_D einzufangen. Der Abstand zwischen p_i und t_D ist mindestens λ, da p_i sonst von links von D eingefangen worden wäre, noch bevor p_R eingefangen wurde. Analog gilt auch im Fall $p_i = p_R$, dass der Abstand zwischen t_d mindestens λ beträgt. Liegen zwischen t_D und p_R keine weiteren Clients, so überschneidet das Intervall (t_D, p_i) mit Länge $\geq \lambda$ kein J_i und der Beweis ist geglückt.

Zu betrachten bleibt also der Fall, in dem zwischen p_i und t_D weitere Clients liegen. Sei p_{i-1} der Client direkt links von p_i. Angenommen, der Abstand zwischen p_{i-1} und p_i beträgt mindestens λ. Sollte p_{i-1} von links gefangen worden sein, enthält das Intervall (p_{i-1}, p_i) keinen Client und der Beweis ist ebenfalls geglückt. Sollte p_{i-1} von rechts eingefangen worden sein, so muss p_{i-1} von einen Server t_k, der zwischen p_{i-1} und p_i liegt eingefangen worden sein, da p_{i-1} gemäß der Definition von p_i nicht vom selben Kreis wir p_i eingefangen worden sein kann. Dann muss der Abstand zwischen p_i und t_k mindestens λ betragen. Andernfalls wäre p_i bereits von links von t_k eingefangen worden, noch bevor p_R eingefangen wurde.

Es bleibt also nur noch der Fall, in dem der Abstand zwischen p_{i-1} und p_i kleiner als λ ist, zu betrachten. Offensichtlich kann dann p_{i-1} nicht vor p_R eingefangen worden sein, da ansonsten p_i sofort vom gleichen Kreis wie p_i eingefangen worden wäre. Dies widerspricht jedoch der Annahme, dass p_i von D_R und p_{i-1} von einem anderen Kreis eingefangen wurden. Desweiteren kann kein Server zwischen p_{i-1} und p_i liegen, da dieser sonst p_i eingefangen hätte. Da p_i der letzte Client ist, der von D_R aktiv von rechts eingefangen wurde und zwischen p_{i-1} und p_i kein Server liegt, muss p_{i-1} also von links eingefangen worden sein. Da p_{i-1} vor p_R eingefangen wurde, muss links von p_i folglich mindestens ein Intervall J_{l_i} mit Länge $\geq \lambda$ liegen, welches zum selben Kreis gehören, der auch p_{i-1} eingefangen hat und die platziert wurden, bevor p_{i-1} eingefangen wurde. Außerdem liegen rechts von p_i Intervalle J_{r_j} mit Länge von mindestens λ, die zu D_R gehören und platziert wurden, bevor p_i eingefangen wurde. Eines dieser Intervalle (möglicherweise das Einzige) ist J_R. Nun gibt es zwei mögliche Fälle: Wird das letzte J_{l_i} platziert, bevor dass letzte J_{r_j} platziert wird, so wird p_{i-1} vor p_i eingefangen. Anschließend betragen die Kosten für das einfangen von p_i von links $d(p_{i-1}, p_i) < \lambda$ und bevor p_i von rechts eingefangen werden kann, muss noch ein J_{r_j} mit einer Länge $\geq \lambda$ platziert werden. Also wird p_i von links eingefangen, was der Definition von p_i widerspricht. Analog führt der zweite Fall, bei dem das letzte J_{r_j} vor dem letzten J_{l_i} platziert wird dazu,

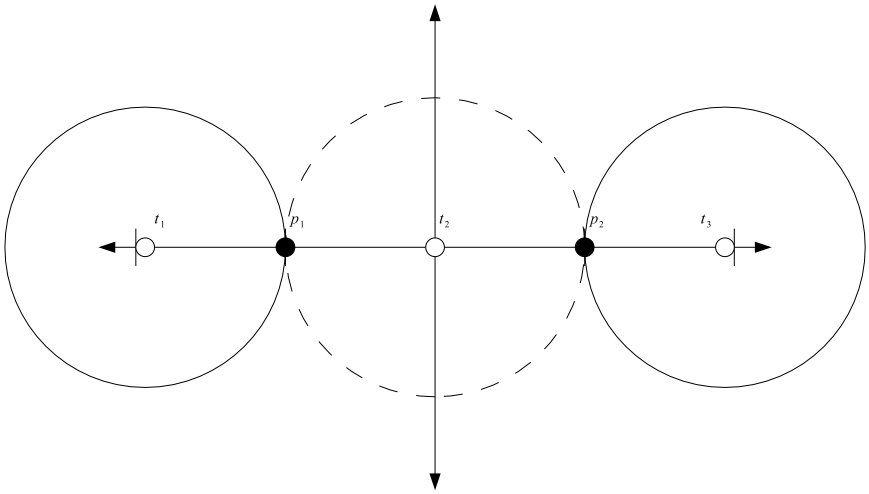


Abbildung 3: Eine Situation, in der die vom GG Algorithmus berechnete Lösung (durchgezeichnet) beinahe die doppelte Kosten der optimalen Lösung (gestrichelt) hat

dass p_{i-1} von rechts eingefangen wird, was ebenfalls zu einem Widerspruch führt.

Also führt die Annahme, dass der Abstand zwischen p_{i-1} und p_i kleiner als λ ist, zwangsläufig zu einem Widerspruch. Folglich liegt links von p_i in dem Intervall $[p_i - \lambda, p_i]$ kein J_k und die Voraussetzungen für die Argumentation, die zu einer oberen Schranke von 2 für die Approximationsgüte führten, sind gegeben.

Eine Situation, die belegt, dass 2 auch Untere Schranke der Approximationsgüte ist, wird in Abbildung 3 dargestellt. Zwei Clients befinden sich bei -1 und 1. Server befinden sich an den Positionen $-2 + \epsilon$, 0 und $2 - \epsilon$. GG berechnet die mit durchgezeichneten Kreisen dargestellte Lösung: Kreise mit Radius $1 - \epsilon$ werden bei $-2 + \epsilon$ und $2 - \epsilon$ platziert. Die Kosten der Lösung betragen $2 - 2 \cdot \epsilon$. Die gestrichelt dargestellte, optimale Lösung, bei der ein Kreis mit Radius 1 bei 0 platziert wird, hat lediglich Kosten von 1. Für beliebig kleine ϵ kommen die Kosten der von GG berechneten Lösung beliebig nah an die doppelten Kosten der optimalen Lösung heran. □

Wie bereits angedeutet, kann der Beweis auf Lemma 1 übertragen werden. Dabei gelten die meisten Argumente unmodifiziert auch für den CGG-Algorithmus. Ordnet man erneut jedem Kreis der optimalen Lösung die J_i der überdeckten Clients zu, so kann erneut nur je ein J_i auf jeder Seite herausragen. Erneut können J_R und J_L lediglich um den Radius des Kreises herausragen. Auch die Argumentation, dass rechts des Zentrums des Kreises ein Bereich der Länge λ frei sein muss, erfolgt analog. Aufgrund des asymetrischen Vorgehens des GGC, lässt sich dieses Argument jedoch nicht auf

die linke Hälfte des Kreises übertragen. Da der Bereich zwischen J_L und dem Zentrum des Kreises erst nach der Platzierung von J_L bearbeitet wird, haben hier platzierte J_i keinen Einfluss auf die Platzierung von J_L. Folglich kann die linke Hälfte nahezu komplett von J_i überdeckt werden. Als Summe der Längen der einem Kreis D aus der optimalen Lösung zugeordneten J_i erhalten wir also $2 \cdot Radius(D) + LängeJ_L \leq 3 \cdot Radius(D)$ und die berechnete Lösung hat maximal die dreifachen Kosten der optimalen Lösung.

3 NP-Härte des zweidimensionalen, diskreten Problems bei $\alpha > 1$

In diesem Abschnitt wird folgendes Problem betrachtet: Gegeben sei eine Menge von Clientpositionen $Y = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^2$ sowie eine Menge möglicher Serverpositionen $X = \{t_1, \ldots, t_m\} \subseteq \mathbb{R}^2$. Anders als in den vorherigen Abschnitten wird also eine 2-dimensionale Variante des Problems betrachtet. Gesucht wird eine Überdeckung von Y durch Kreise, deren Mittelpunkte in X liegen, und die minimale Gesamtkosten hat. Dabei sind die Kosten eines Kreises mit Radius r $f(r) = r^\alpha$ mit $\alpha > 1$. α ist dabei fest, also *nicht* Teil der Eingabe. Es wird die euklidische Metrik verwendet.

Über dieses Problem kann folgende Aussage getroffen werden:

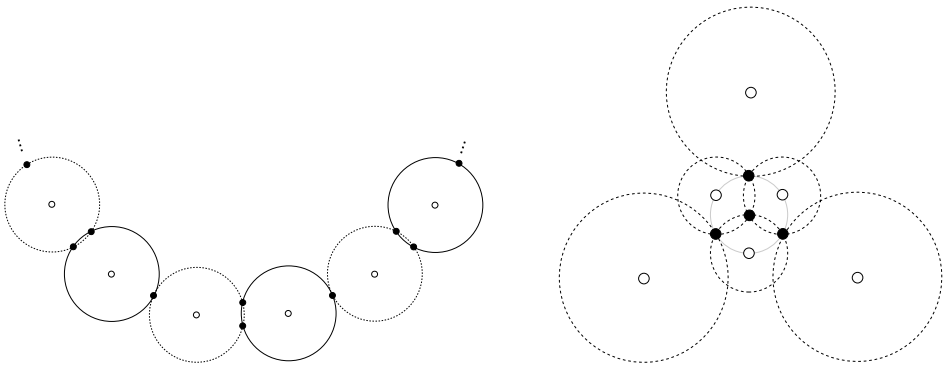
Theorem 3 *Für beliebiges, festes $\alpha > 1$ seien die Kosten eines Kreises mit Radius r als $f(r) = r^\alpha$ definiert. Dann ist es es NP-hart, zu entscheiden, ob die minimalen Kosten einer Überdeckung von n Clientpositionen durch Kreise, deren Zentren Element einer vorgegebenen, diskreten Menge mit m Elementen sind, einen bestimmten Wert unterschreiten.*

Beweis. *(Skizze)*
Die Idee des Beweises ist, das Planar-3Sat-Problem, von dem bereits bekannt ist, dass es NP-hart ist, zu reduzieren. Bei Planar-3Sat handelt es sich um eine Variation des 3SAT-Problems. Beim 3SAT-Problem ist eine Formel in konjunktiver Normalform, bei der jede Klausel aus maximal 3 Literalen besteht, gegeben, und es soll festgestellt werden, ob eine Belegung existiert, die die Formel erfüllt.

Für eine Instanz des 3SAT-Problems wird ein der folgende, ungerichtete Graph erzeugt:

- Für jede Variable und jede Klausel wird ein Knoten erzeugt.
- Je eine Kante wird zwischen jedem Klausel-Knoten und jeder in der Klausel vorkommenden Variable platziert.

Eine Instanz des 3Sat-Problems ist Instanz des Planar-3Sat, genau dann, wenn der zugehörige Graph planar ist.



(a) Ausschnitt eines Variablen-Bereichs. Um den gesamten Variablen-Bereich darzustellen, muss die dargestellte Kette zu einem Kreis vervollständigt werden.

(b) Ein Klausel-Bereich

Abbildung 4: Bei der Reduktion des Planar-3Sat vorgenommene Ersetzungen

Sei I eine Instanz der Planar-3Sat-Problems und G der zugehörigen planare Graphs. Dann wird eine kreuzungsfreie geometrische Realisation von G in $\mathbb{R}^2$ berechnet. Dies ist mit ganzzahligen Koordinaten, die durch ein Polynom über die Anzahl der Knoten und Kanten in G beschränkt sind, möglich [2]. jeder Variablenknoten in G durch eine grob im Kreis angeordnete Menge von Client- und Serverpositionen ersetzt. Dabei wird das in Abbildung 4(a) angedeutete Schema verwendet: Entlang des Randes eines gedachten Kreises werden immer abwechselnd ein Server und ein oder zwei Clients platziert. Dabei wird für jede Variable eine gerade Anzahl von Serverpositionen platziert. Zur Überdeckung der Clients kann bei jeder zweiten Serverposition ein Kreis platziert werden, der die drei benachbarten Clients überdeckt. Die Abstände der Client- und Serverpositionen werden so gewählt, dass alle diese Kreise gleich groß sind. Dann gibt es, wie man sich anhand von Abbildung 4(a) verdeutlichen kann, genau zwei mögliche minimale Überdeckungenen des Variablen-Bereichs: bei einer werden die gestrichelten Kreise, bei der anderen die durchgezeichneten Kreise platziert. Größere Teile können nicht Teil der Lösung sein, da $\alpha > 1$ gilt. Jeder der beiden möglichen minimalen Überdeckungen wird eine Belegung der zum Bereich gehörenden Variable zugegordnet. Im Folgenden wird angenommen, dass die durchgezeichneten Kreise einer Belegung mit „Wahr“, und die gestrichelten einer Belegung mit „Falsch“ entsprechen.

Neben den Variablen-Bereichen wird auch jede Klausel durch ein spezielles Konstrukt ersetzt. Dieses wird in Abbildung 4(b) dargestellt. In jedem Klausel-Bereich liegen vier Clients und fünf Serverpositionen. Isoliert betrachtet können die Clients des Klausel-Bereichs optimal überdeckt wer-

den, indem zwei der drei kleinen angedeuteten Kreise gesetzt werden. Ist allerdings einer der großen Kreise bereits gesetzt, so überdeckt er einen der Clients und die restlichen können durch den dem großen Kreis gegenüberliegenden kleinen Kreis überdeckt werden. Andererseits muss zwangsläufig ein kleiner Kreis zur Überdeckung des mittleren Clients platziert werden - selbst wenn alle großen Kreise gesetzt sind. Die Situation lässt sich also wie folgt zusammenfassen:

- Ist keiner der großen Kreise gesetzt, so müssen zwei der drei kleinen Kreise gesetzt werden. Welche beiden Kreise gesetzt werden, kann beliebig gewählt werden.
- Ist mindestens einer der großen Kreise gesetzt, so muss genau ein kleiner Kreis gesetzt werden. Welcher dies ist, hängt davon ab, welche großen Kreise gesetzt sind.

Es gibt natürlich auch noch Möglichkeiten, die Clients mit anderen (größeren) Kreisen, als den dargestellten zu überdecken. So könnte beispielsweise statt zwei kleinen Kreisen mit Radius r ein Kreis mit Radius $2r$ um einen der inneren Serverpositionen platziert werden. Dabei würden allerdings wegen $\alpha > 1$ Kosten von $(2r)^\alpha = 2^\alpha r^\alpha > 2r^\alpha$, also höhere Kosten entstehen. Aus ähnlichen Gründen wird es nie optimal sein, einen der großen Kreise zu vergrößern, statt einen kleinen zu platzieren.

Jeder große Kreis eines Klausel-Bereichs gehört zu einem Literal in der Formel und wird mit einer Kette so mit dem zugehörigen Variablen-Bereich verbunden. Tritt die Variable in der Formel nicht negiert auf, so geschieht die Verbindung so, dass der große Kreis im Klausel-Bereich genau dann gesetzt werden muss, wenn die Variable mit „Wahr“ belegt wird. Analog werden große Kreise, die zu einer negierten Variable gehören so verbunden, dass sie gesetzt werden müssen, falls die Variable mit „Falsch“ belegt wird. Dementsprechend wird mindestens ein großer Kreis des Klausel-Bereichs gesetzt, wenn mindestens ein Literal den Wert „Wahr“ ergibt, also wenn die zugehörige Klausel erfüllt wird. Dementsprechend ist die betrachtete 3SAT-Formel genau dann erfüllbar, wenn es eine Überdeckung aller Clients gibt, bei der in jedem Klausel-Bereich nur ein kleiner Kreis gesetzt wird. Wie am Ende des Beweises gezeigt wird, ist eine solche Überdeckung, falls sie existiert, zwangsläufig minimal.

Zunächst wird jedoch gezeigt, wie die Ketten zwischen den Klausel-Bereichen und den Variablen-Bereichen konstruiert werden. Ihre wichtigste Eigenschaft ist, dass sie die Belegung der Variablen propagieren. Desweiteren sollen die Kosten zur Überdeckung einer Kette unabhängig von der Belegung der zugehörigen Variablen konstant sein. Abbildung 5 skizziert die Verbindung zwischen einem Variablen-Bereich und einem Klausel-Bereich. Sind bei dem Variablen-Bereich die durchgezeichneten Kreise gesetzt, so müssen auch auf der Kette die durchgezeichneten Kreise platziert werden. Folglich

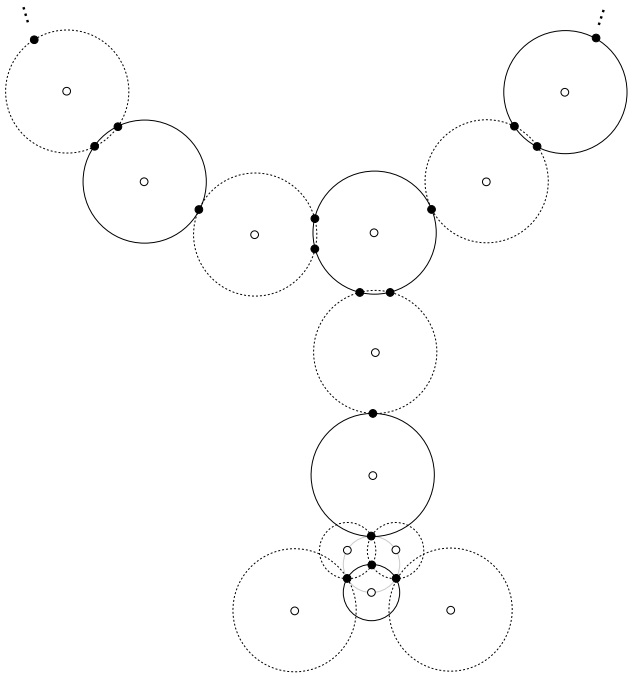


Abbildung 5: Die Verbindung zwischen eine Variablen-Bereich und einem Klausel-Bereich.

wird der obere Kreis des Klausel-Bereich gesetzt und die verbleibenden Clients der Klausel können durch setzen des unteren kleinen Kreises überdeckt werden.

Angenommen, das Setzen der durchgezeichneten Kreise entspricht einer Belegung der Variable v mit dem Wert „Wahr“. Dann entspricht die Kette einem nicht-negierten Vorkommen der Variable v in der zu dem Klausel-Bereich gehörenden Klausel. Würde v stattdessen negiert vorkommen, so müsste die Kette an einem der gestrichelten Kreise platziert werden.

Die Ketten können beliebig verlängert werden, sofern dabei eine gerade Anzahl von Serverpositionen hinzugefügt wird. Da die Variablen-Bereiche und die Klausel-Bereiche Knoten und die Ketten Kanten eines planaren Graphen entsprechen, können die Ketten immer ohne Überkreuzungen platziert werden.

Nun werden die Kosten einer optimalen Überdeckung abgeschätzt. Da jeder Variablen-Bereich eine gerade Anzahl von Serverpositionen besitzt, wird unabhängig davon wie die zugehörige Variable belegt wird, um die Hälfte der Serverpositionen Kreise gelegt. Da diese Kreise außerdem alle den selben Radius haben, sind die Kosten der Variablen-Bereiche unabhängig von der Variablenbelegung. Die gleichen Argumente gelten für die Kosten der Ketten. Existiert eine erfüllende Belegung für das Planar-3Sat-Problem,

so reicht es in jedem Klausel-Bereich einen kleinen Kreis zu platzieren. Sei s_v die Anzahl der Serverpositionen in Variablen-Bereichn, s_K die Anzahl der Serverpositionen in Ketten und k die Anzahl der Klauseln. Desweiteren seien c_k die Kosten eines kleinen und c_G die Kosten eines großen Kreises. Dann sind die minimalen Gesamtkosten, wenn eine erfüllende Belegung existiert, genau $c_{erf} = \frac{s_v}{2} \cdot c_G + \frac{s_K}{2} \cdot c_G + k \cdot c_k$. Existiert keine erfüllende Belegung, so müssen in mindestens einem Klausel-Bereichzwei kleine Kreise platziert werden. Die Kosten betragen also mindestens $\frac{s_v}{2} \cdot c_G + \frac{s_K}{2} \cdot c_G + k \cdot c_k + c_k = c_{erf} + c_k > c_{erf}$. Ist nun also bekannt, ob eine Überdeckung mit Kosten $\leq c_erf + \frac{r_k}{2}$ existiert, so kann hieraus geschlossen werden, ob eine erfüllende Belegung der Formel existiert.

Um den Beweis abzuschließen, muss allerdings noch gezeigt werden, dass die Reduktion in polynomieller Zeit erfolgen kann. Da die räumliche Anordnung der Variablen-Bereiche und der Klausel-Bereiche der Knotenplatzierung einer kreuzungsfreien Realisierung des Graphs G entspricht, ist zunächst wichtig, dass G so eingebettet wurde, dass alle Komponenten der Knotenpositionen ganzzahlig und durch einen Wert, der polynomiell von der Anzahl der Kanten und Knoten von G abhängt, beschränkt sind. Diese Anzahlen hängen wiederum polynomiell von der Größe der Planar-3Sat-Formel ab. Hieraus lässt sich schließen, dass die maximalen Entfernungen zwischen den Klausel-Bereichen und den Variablen-Bereichen nur polynomiell von der Größe der Eingabe abhängen. Desweiteren ist aufgrund der ganzzahligen Koordinaten der Abstand zwischen benachbarten Variablen-Bereichn und Variablen-Bereichn 1 nicht unterschreitet. Dies ist wichtig, da die Größe der Kreise in Abhängigkeit des verfügbaren Platzes gewählt werden muss.

Es werden für jeden Variablen-Bereich eine von der Anzahl der anzuschließenden Klauseln abhängige Anzahl von Clientpositionen und Serverpositionen benötigt. Für jeden Klausel-Bereich wird eine feste Anzahl benötigt. Desweiteren lässt sich erahnen, dass aufgrund der bereits dargelegten Argumente die Anzahl der nötigen Clientpositionen und Serverpositionen in Ketten beschränkt ist und das hieraus folgt, dass die Reduktion in polynomieller Zeit möglich ist.

Auf einen exakten Beweis dafür, dass die Reduktion in poynomieller Zeit möglich ist, wird hier, wie auch in [2], verzichtet. □

4 Serverpositionen beschränkt auf eine feste, horizontale Gerade

In diesem Szenario können die Serverpositionen beschränkt auf eine Gerade, die ohne Beschränkung der Allgemeinheit die X-Achse ist, frei gewählt werden. Derartige Szenarien können beispielsweise daraus resultieren, dass Server entlang einer Autobahn oder einem Gang in einem Gebäude liegen

müssen. Desweiteren ist die Lösung des Problems eine Voraussetzung für die Lösung eines anderen Problems, bei dem die Server entlang eines Polygonzuges liegen müssen. Dieses Problem wird allerdings im Rahmen dieser Ausarbeitung nicht betrachtet.

4.1 Exakte Lösung

In diesem Abschnitt wird ein Algorithmus der dynamischen Programmierung vorgestellt, der eine Menge von Serverpositionen und zugehörigen Kreisen zur Überdeckung einer Menge von Clients $Y = \{p_1, \ldots, p_n\}$ mit minimalen Kosten berechnet. Der Bequemlichkeit halber wird angenommen, dass die Clients in aufsteigender Folge entlang der X-Achse indiziert sind. Desweiteren wird ohne Beschränkung der Allgemeinheit angenommen, dass alle Clientposition paarweise verschieden sind und oberhalb oder auf der X-Achse liegen. Sollten zwei Clients an der selben Position liegen, kann einer weggelassen werden. Sollte ein Client an der Position (x, y) mit $y < 0$ liegen, so kann er an die Position $(x, -y)$ verschoben werden, da jeder x-Achsen-zentrierte Kreis, der (x, y) überdeckt auch $(x, -y)$ überdeckt und umgekehrt.

Ein Kreis C heißt *festgenagelt*, wenn er der am weitesten links liegende, X-Achsen-zentrierte Kreis mit dem geringsten Radius ist, der eine Menge von Clients aus Y überdeckt. (Wird die L_∞-Metrik verwendet, so sind Kreise Quadrate und es gibt Mengen von Clients, die von mehreren Kreisen des gleichen minimalen Radius überdeckt werden. Um in diesem Fall die Eindeutigkeit des festgenagelten Kreises zu erhalten, wird der festgenagelte Kreis als der am weitesten links liegende Kreis definiert.)

Eine wichtige Beobachtung ist, dass ein festgenagelter Kreis C einer Menge $Y' \subseteq Y$, stets gleichzeitig der festgenagelte Kreis einer ein- oder zweielementigen Menge $Y'' \subseteq Y'$ ist. Liegt nur ein Client p_i auf dem Rand von C - dieser liegt dann zwangsläufig am höchsten Punkt von C - so ist C auch gleichzeitig der festgenagelte Kreis von $\{p_i\}$. Liegen mehrere Clients auf dem Rand des Kreises, so liegt zwangsläufig mindestens einer links und rechts des Zentrums. Lägen alle Clients auf dem Rand auf einer Seite des Zentrums, so könnte man C so in diese Richtung verschieben, das er verkleinert werden kann und dennoch die gleichen Clients wie zuvor überdeckt werden. Man kann also auf dem Rand von C einen Client links des Zentrums und einen rechts davon wählen. Dann ist C ebenfalls festgenagelter Kreis dieser beiden Clients. (Die Argumentation ist nicht gültig für die L_∞-Metrik – wird sie verwendet, so müssen stattdessen ein Client auf dem rechten Rand des Rechtecks und einer auf dem oberen oder linken Rand gewählt werden.) Aus diesen Beobachtung lässt sich ableiten, dass folgende Definition eines festgenagelten Kreises äquivalent ist: Ein Kreis C heißt festgenagelt, wenn er der kleinste, am weitesten links liegende Kreis ist, auf dessen Rand ein bestimmter Client oder ein bestimmtes Paar von Clients liegt. Folglich gibt

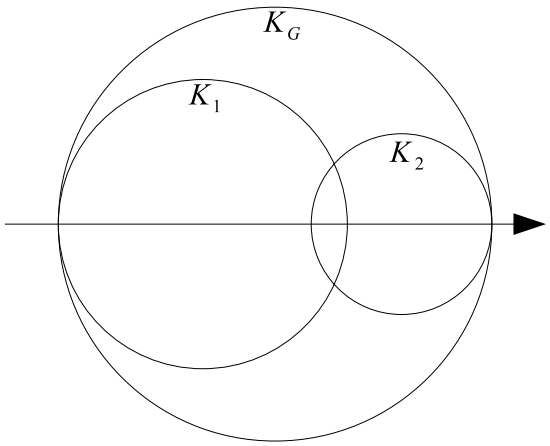


Abbildung 6: K_G überdeckt $K_1 \cup K_2$ und hat bei $\alpha = 1$ geringere Kosten

es maximal $n(n-1) + n \in O(n^2)$ festgenagelte Kreise.

Sofern die Kostenfunktion f monoton steigt, gibt es eine optimale Lösung des Problems, die nur aus festgenagelten Kreisen besteht. Jede optimale Lösung, die nicht nur aus festgenagelten Kreisen besteht, lässt sich in eine optimale Lösung aus festgenagelten Kreisen überführen, in dem jeder Kreis C durch den festgenagelten Kreis zu den Clients, die C überdeckt, ersetzt wird. Dabei wachsen die Kosten der Lösung nicht, da kein Kreis eine Menge von Clients mir geringeren Kosten als denen des festgenagelten Kreises überdecken kann.

4.1.1 Lineare Kosten

Zunächst wird eine lineare Kostenfunktion betrachtet. In diesem Fall überschneiden sich die Kreise der optimalen Lösung nicht, da für jedes Paar überschneidender Kreise ein größerer Kreis existiert, der die beiden kleineren enthält und dessen Radius kleiner ist, als die Summe der Radien der kleinen Kreise. Dies wird auch in Abbildung 6 verdeutlicht.

Ein Algorithmus der dynamischen Programmierung, der eine optimale Lösung des Problems berechnet, ist Algorithmus 4.1. Dabei ist der Besitzer eines festgenagelten Kreises der am weitesten rechts in ihm liegende Client. Der Algorithmus verarbeitet die Kreise von links nach rechts. Betrachtet werden in jeder Iteration alle festgenagelten Kreise, deren Besitzer der aktuell betrachtete Client ist. Alle festgenagelten Kreise, oberhalb denen Clients liegen, können nicht Teil der optimalen Lösung sein, da ein weiterer Kreis zur Überdeckung der oberhalb liegenden Kreise benötigt würde, und dieser den ersten Kreis schneiden würde. Jeder Client p_i ist zwangsläufig Besitzer mindestens eines festgenagelten Kreises, oberhalb dessen keine Clients liegen, nämlich dem zu $\{p_1, \ldots, p_i\}$.

Algorithmus 4.1 MinSumOfRadiusCircleConver(Y)

```
for jeden festgenagelten Kreis do
  Finde den am weitesten rechts und links liegenden Punkt im Kreis
end for
Kosten[0] := 0;
for i = 1 to n do
  Kosten[i] := ∞;
  for jeden festgenagelten Kreis K dessen Besitzer p_i ist do
    if kein Punkt in P liegt direkt oberhalb von K then
      p_j := am weitesten links liegender Punkt in K;
      Kosten[i] := min{Kosten[i], Kosten[j − 1] + radius(K)};
    end if
  end for
end for
return Kosten[n];
```

Der Algorithmus berechnet für jeden Client p_i Überdeckungen der Clients $\{p_1,\ldots,p_i\}$. Diese bestehen aus einem festgenagelten Kreis, der Clients $\{p_j,\ldots,p_i\}$ überdeckt, sowie der optimalen Überdeckung der Clients $\{p_1,\ldots,p_{j-1}\}$. Folglich sind $Kosten[n]$ die minimalen Kosten der Überdeckung von $p_1,\ldots,p_n$.

Die triviale Implementierung des Algorithmus benötigt $O(n^3)$ Zeit, da für jeden der $O(n^2)$ festgenagelten Kreise bestimmt werden muss, ob einer der n Clients oberhalb des Kreises liegt, und welcher der n Clients am weitesten links und rechts innerhalb des Kreises liegt. Mit einigen zusätzlichen Überlegungen lässt sich die Zeitkomplexität jedoch um beinahe einen linearen Faktor verringert werden.

Am einfachsten ist die Verbesserung der Laufzeit, wenn die L_∞-Metrik verwendet wird. Nun sind Kreise Quadrate und jeder Client p_i ist Besitzer von exakt i festgenagelten Quadraten: einem mit p_i in der rechten, oberen Ecke und einem für jedes $j < i$ mit p_i auf dem rechten Rand und p_j auf dem linken oder oberen Rand. Folglich können diese Quadrate in $O(i)$ berechnet werden. Sortiert man die Clients zu Beginn (in $O(n \log n)$ Zeit) nach der x-Koordinate, so kann außerdem der für jedes festgenagelte Quadrat der am weitesten links liegende Client mit binärer Suche gefunden werden. Der am weitesten rechts liegende Client ist zwangsläufig p_i. Also können alle festgenagelten Quadrate, deren Besitzer p_i ist sowie die am weitesten links und rechts darin liegenden Clients in $O(i \log i)$ Zeit berechnet werden. Insgesamt muss dies für n Clients durchgeführt werden. Also können also alle festgenagelten Quadrate inklusive den Besitzern und den am weitesten rechts und links liegenden Clients in $O(n^2 \log n)$ berechnet werden.

Um zu testen, ob ein Client oberhalb eines Quadrats mit Zentrum bei x und Radius r liegt, muss lediglich getestet werden, ob Clients oberhalb

des Intervalls $[x-r, x+r]$ auf der x-Achse liegen, deren Höhe größer als r ist. Hierfür eignet sich die Datenstruktur „Priority-Search-Tree“ besonders gut. Ein Priority-Search-Tree ist ähnlich aufgebaut, wie ein Maximums-Heap bezüglich der Höhe der Clients. Allerdings werden die Clients so auf die Teilbäume unter der Wurzel verteilt, dass im linken Teilbaum die linke Hälfte der Clients liegt im rechten die rechte. Auf tieferen Ebenen wird analog verfahren. Nun kann wie folgt getestet werden, ob ein Client mit einer bestimmten Mindesthöhe über einem Intervall liegt: Zunächst wird getestet, ob die Wurzel die Bedingung erfüllt. Ist dies nicht der Fall, so wird getestet, ob die Wurzel zumindest die Mindesthöhe besitzt. Ist dies ebenfalls nicht der Fall, so gibt es keinen Client, der die Mindesthöhe besitzt. Andernfalls wird getestet, ob das zu testende Intervall das Intervall, in dem die Clients des linken Teilbaums liegen, schneidet. Ist dies der Fall, wird rekursiv getestet, ob im linken Teilbaum ein Client liegt, der die Bedingungen erfüllt. Mit dem rechten Teilbaum wird analog verfahren. Der Priority Search Tree kann in $O(n \log n)$ erstellt werden. Die Suche nach einem Client, der eine bestimmte Mindesthöhe hat und über einem bestimmten Intervall liegt, benötigt $O(\log n)$ Zeit. Folglich benötigt der Algorithmus insgesamt $O(n^2 \log n)$ Zeit.

Etwas komplizierter ist die Laufzeitreduktion für eine beliebig L_p-Metrik, mit $p < \infty$. Zunächst wird wieder das Berechnen der Extrempunkte innerhalb der festgenagelten Kreise betrachtet. Hierfür wird die folgende Dualisierung verwendet: Sei C ein Kreis mit Mittelpunkt bei $(t, 0)$ und Radius r. Dann ist $C^* = (t, r)$ der duale Punkt zu C. C^* ist also der Apex (der höchste Punkt) des Kreises. Sei p ein Client. Dann ist p^* die Menge der dualen Punkte aller Kreise, auf deren Rand p liegt: $p^* = \{C^* | p \text{ liegt auf dem Rand von } C\}$. Folglich sind die p_i^* x-monotone Kurven und das (globale) Minimum von p_i^* liegt bei p_i. Links von p_i fällt p_i^* monoton. Rechts von p_i steigt p_i^* monoton. Wie die p_i^* exakt aussehen, hängt von der verwendeten Metrik ab: verwendet man die L_2-Metrik, so sind die p_i^* Hyperbeln. Unter Verwendung der L_1-Metrik bestehen sie aus zwei Halbgeraden, die sich in p_i berühren. Für ein beliebiges Paar von Clients p_i und p_j mit $i \neq j$ ist der Schnittpunkt zwischen p_i^* und p_j^* gleich C^*, wobei C der festgenagelte Kreis zu $\{p_i, p_j\}$ ist. Da dieser Kreis eindeutig definiert ist, können p_i^* und p_j^* sich auch nur einmal schneiden (eine Ausnahme besteht, wenn die L_1-Metrik verwendet wird, dann existieren potentiell unendlich viele Schnittpunkte, wenn zwei Halbgeraden der p^* aufeinander liegen). Eine weitere wichtige Beobachtung ist, dass ein Kreis C den Client p genau dann enthält, wenn C^* oberhalb von oder auf p^* liegt.

Die p^* teilen den $\mathbb{R}^2$ in Sektoren auf. Diese Aufteilung wird das Arrangement von $Y^* = \{p^* | p \in Y\}$ genannt. Da je zwei p^* sich maximal einmal schneiden, kann das Arrangement in $O(n^2)$ berechnet werden. Das Arrangement ist die Einbettung eines planaren Graphs. Sei C ein festgenagelter Kreis. Wie bereits zuvor argumentiert wurde, liegen entweder auf dem Rand von C zwei oder im Apex von C ein Client. Folglich ist C^* entweder das

Minimum eines p^* (also p), oder der Schnittpunkt zwischen zwei p^*, also ein Vertex des Arrangements. Wurde das Arrangement berechnet, kann anschließend für jeden Vertex und jeden Client (also für jeden dualen Punkt eines festgenagelten Kreises) berechnet werden, welche p^* unterhalb des jeweiligen Punktes liegen (also welche Clients von dem festgenagelten Kreis überdeckt werden). Anschließend muss festgestellt werden, welches der am weitesten links beziehungsweise rechts liegende unter diesen Clients ist. Dies kann wie folgt insgesamt in $O(n^2)$ für alle Vertices berechnet werden:

Zunächst wird der Teil des Arrangements, der außerhalb eines rechteckigen Bereiches R, der alle Vertices (also Schnittpunkte und Duale der Clientpositionen) des Arrangements enthält, Abgeschnitten. Die Schnitte des Arrangements mit dem Rand des rechteckigen Bereichs R werden als Knoten und der Rand selbst als Kanten zum Arrangement hinzu genommen. Der resultierende Graph G ist also kreuzungsfrei und zusammenhängend. Da außerdem jede p^* den Rand von R höchstens zwei mal schneidet, liegen die Anzahlen der Knoten und Kanten von G immer noch in $O(n^2)$.

Es soll nun also für jeden Knoten von G berechnet werden, welches p^* am weitesten links unterhalb des Knotens liegt (genauer gesagt welches p^* unterhalb das Dual des am weitesten links liegenden Clients ist). Die Berechnung der am weitesten rechts liegenden p^* erfolgt analog. Zunächst wird die linke Kante von R von unten nach oben verfolgt. Dabei wird verfolgt, welches das am weitesten links liegende p^* unterhalb der aktuell betrachteten Position auf der linken Kante von R ist. Ein Knoten auf der linken Kante von R, entspricht dem Schnittpunkt des Randes von R mit einem p_i^*. Liegt der zugehörige Client p_i weiter links, als der zuvor am weitesten links liegende Client, so ist p_i oberhalb des Knotens der neue am weitesten links liegende Client. Nach dem Verfolgen der Kante ist also für alle Knoten auf der linken Kante von R bekannt, welches das am weitest links liegende p^* unterhalb ist. Diese Information wird nun entlang der Kanten von G propagiert. Welches p_j^* das am weitesten links liegende Unterhalb eines p_i^* ist, kann sich nur an Schnittpunkten zwischen p_i^* und einem p_k^* ändern. Diese Schnittpunkte sind Knoten in G. Wird also beim Verfolgen eines p_i^* einer Schnittpunkt mit p_j^* getroffen ist zwischen zwei Fällen zu unterscheiden:

- Traversiert p_j^* p_i^* von unten nach oben, so muss getestet werden, ob p_j^* zuvor das am weitesten links liegende p^* unterhalb von p_i^* war. Ist dies der Fall, so ist das neue am weitesten unterhalb liegende p^* unterhalb von p_i^* das unterhalb von p_j^*.

- Traversiert p_j^* p_i^* von oben nach unten, so muss getestet werden, ob p_j^* weiter links liegt, als des zuvor am weitesten links liegende p^* unter p_i^*. In diesem Fall ist p_j^* das neue am weitesten links liegende p^*.

Unter Zuhilfenahme dieser beiden Regeln lässt sich also in $O(n^2)$ berechnen, welches die am weitesten links liegenden p^* unterhalb der Knoten von G

sind. Um daraus abzuleiten, welches die am weitesten links liegenden Clients innerhalb eines Kreises C sind, müssen allerdings noch die (maximal 2) p^*, die durch C^* verlaufen betrachtet werden.

Nun wird eine Datenstruktur vorgestellt, die erlaubt, effizient herauszufinden, ob Clients oberhalb eines Kreises liegen. Zunächst werden alle Clients in einem Blattsuchbaum nach der X-Komponente ihrer Position sortiert. Hierfür wird $O(n \log n)$ Zeit benötigt. Jeder innere Knoten v des Suchbaumes korrespondiert dann zu einer Menge $Y_v \subseteq Y$ von Clients sowie zu einem achsenparallelen, vertikalen Streifen s_v, der alle Clients aus K_v enthält. Für jeden inneren Knoten wird die x-Achse in Intervalle aufgeteilt, indem sie mit dem Furthest-Point-Voronoi-Diagramm zu Y_v geschnitten wird. Im Gegensatz zum klassischen Voronoi-Diagramm enthält das für einen Client $p \in Y_v$ eine Zelle von Punkten, die weiter von p entfernt sind, als von jedem anderen Punkt in Y_v (nicht für jeden Client existiert eine solche Zelle, aber jeder Punkt des $\mathbb{R}^2$ liegt in solch einer Zelle). Für jeden inneren Knoten v lassen sich die resultierenden X-Achsen-Abschnitte in $O(|Y_v| \log |Y_v|)$ berechnen. Wegen $|Y_v| \leq 1/2^{Tiefe(v)} \leq \lceil n/2^{Tiefe(v)} \rceil$ und weil mit Tiefe k 2^k innere Knoten existieren und die maximale Tiefe $O(\log n)$ ist, sind die Gesamtkosten der Erstellung der Datenstruktur $O(\sum_{k=1}^{c \cdot \log n} 2^k \frac{n}{2^k} \log \frac{n}{2^k}) \subseteq O(n(\log n)^2)$.

Sei t ein Punkt, der in einem x-Achsen-Abschnitt liegt, der die Furthest-Point-Voronoi-Diagramm-Zelle von $p \in Y_v$ schneidet, dann ist t weiter von p entfernt, als von jedem anderen Client aus Y_v. Folglich enthält ein Kreis um t, der p enthält, ganz Y_v. Diese Beobachtung lässt sich nun nutzen, um zu testen, ob ein Client oberhalb eines Kreises C liegt: Zunächst werden im Suchbaum die $O(\log n)$ inneren Knoten gesucht, unter denen genau die Clients liegen, die über dem von C überdeckten x-Achsen-Intervall liegen. Für jeden gefundenen inneren Knoten v wird dann in $O(\log n)$ betrachtet, in welchem x-Achsen-Abschnitt das Zentrum von C liegt. Hieraus lässt sich der am weitesten vom Zentrum von C entfernte Client, der im Suchbaum unter v liegt, ablesen. Ist dieser Knoten in C enthalten, so sind alle unter v liegenden Knoten in C enthalten. Die Kosten der Anfrage sind also $O((\log n)^2)$, können jedoch unter Verwendung von Fractional Cascading auf $O(\log n)$ reduziert werden [2]. Die Gesamtkosten sind folglich $O(n^2 \log n)$.

Theorem 4 *Für jede beliebige, feste L_p-Metrik kann eine Überdeckung von n Clients in der Ebene durch x-Achsen-zentrierte Kreise mit minimaler Summe der Radien in $O(n^2 \log n)$ berechnet werden.*

4.1.2 Superlineare Kosten

Ein ähnlicher Algorithmus berechnet die Kosten einer optimalen Überdeckung für jede beliebige monoton steigende Kostenfunktion, insbesondere also auch für superlineare Kosten. Obwohl der Algorithmus mit jeder L_p-Metrik funktioniert, wird zunächst von endlichem p ausgegangen. Das Ergebnis

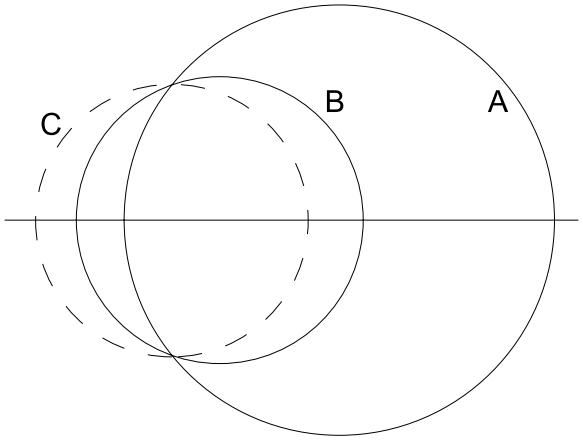
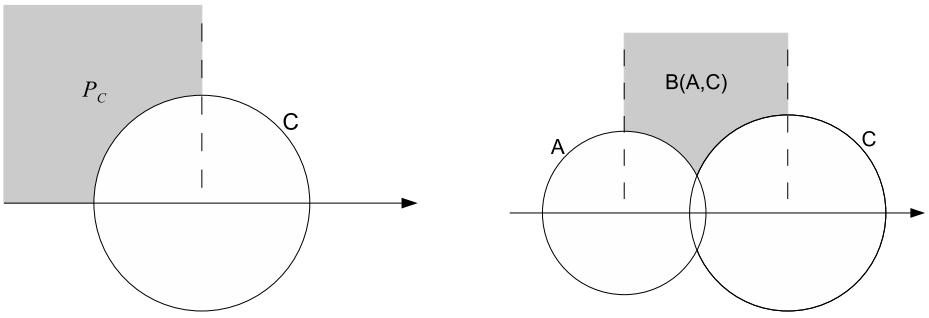


Abbildung 7: C überdeckt $B \setminus A$ und hat geringere Kosten als B.



(a) Alle Clients aus Y_C liegen in P_C, dem grauen Bereich

(b) $B(A, C)$ (grau dargestellt).

Abbildung 8: P_C, Y_C und $B(A, C)$

wird anschließend auf die L_∞-Metrik übertragen.

Anders als im linearen Fall können zwei Kreise der optimalen Lösung bei sich superlinearen Kosten durchaus überschneiden - allerdings nur in begrenztem Maß. Offensichtlich kann keine Lösung, bei der ein Kreis einen anderen enthält, optimal sein. Desweiteren kann eine Lösung, bei der ein Kreis den Apex (den höchsten Punkt) eines anderen Kreises enthält, nicht optimal sein. Diese Situation wird in Abbildung 7 dargestellt. A enthält den Apex von B. Der gestrichelt dargestellte Kreis C überdeckt den Halbmond $B \setminus A$ und hat einen kleineren Radius als B.

Für jeden festgenagelten Kreis C sei die Menge $P_C \subseteq \mathbb{R}^2$ definiert als der nach links und oben offene Bereich, der links des Zentrums, außerhalb von C und oberhalb der x-Achse liegt. Zur Verdeutlichung dient die Abbildung 8(a). Desweiteren sei für festgenagelte Kreise C und A die Region $B(A, C)$ definiert als $P_C \setminus P_A$. Der Bereich, in dem $B(A, C)$ liegt, wird in Abbildung 8(b) verdeutlicht. $Y_C \subseteq Y$ seien die Clients, die in P_C liegen.

Der Algorithmus basiert auf der folgenden Beobachtung: sei R der am weitesten rechts liegende Kreis in einer optimalen Überdeckung O. Dann liegen alle Clients entweder in Y_R oder in R. Sollte ein Client nicht in Y_R oder R liegen, also rechts des Zentrums von R und außerhalb von R, so müsste er von einem Kreis, dessen Zentrum links des Zentrums von R liegt, überdeckt werden. Ein solcher Kreis würde den Apex von R enthalten. Wie bereits betrachtet kann eine solche Überdeckung nicht optimal sein.

Also besteht O aus R und einer Überdeckung O' von Y_R. Sei R_2 der am weitesten rechts liegende Kreis in O', also der zweite Kreis von rechts in O. Dann muss R_2 aufgrund der Optimalität von O folgende Bedingungen erfüllen:

- R_2 enthält den Apex von R nicht. Die Gründe hierfür wurden bereits dargelegt.
- R enthält den Apex von R_2 nicht.
- $B(R_2, R)$ ist leer. Läge in $B(R_2, R)$ ein Client, so müsste er von einem Kreis, dessen Zentrum links des Zentrums von R_2 liegt, überdeckt werden. Jeder solche Kreis, der in $B(R_2, R)$ vordringt, enthält zwangsläufig den Apex von R_2 und kann deshalb nicht Teil einer optimalen Überdeckung sein.

Da alle Überdeckungen von Y_R, die diese drei Bedingungen erfüllen, potentiell Teil einer optimalen Überdeckung sind, ist O' die Überdeckung von Y_C, die die Bedingungen erfüllt und außerdem minimale Kosten unter allen solchen Überdeckungen hat. Analog besteht O' aus R_2 und einer minimalen Überdeckung von Y_{R_2}, die ebenfalls den genannten Bedingungen gehorcht. Aus dieser Beobachtung lässt sich der Dynamische-Programmierung-Algorithmus MinSuperlinearCostCircleCover (Algorithmus 4.2) ableiten:

Dabei ist p die Anzahl der festgenagelten Kreise. Es werden die folgenden vereinfachenden Notationen verwendet: Y_0 statt $\emptyset$, Y_{p+1} statt Y und Y_i statt Y_{C_i}.

Die festgenagelten Kreise werden zunächst nach der X-Koordinate ihres Mittelpunktes sortiert und dann von links nach rechts verarbeitet. Für jeden festgenagelten Kreis C_i wird $Cost[i]$ - die Kosten einer minimalen Überdeckung von Y_i, die den oben genannten Bedingungen genügt - berechnet. Es enthält also kein Kreis den Apex eines anderen und für benachbarte Kreise C_a und C_b ist $B(C_a, C_b)$ leer. Dazu werden alle Kreise C_j deren Zentren links des Zentrums von p_i liegen betrachtet. Genügt ein Paar aus C_j und C_i den genannten Bedingungen, so wird aus der bereits gefunden minimalen Überdeckung für Y_{C_j} durch Hinzunahme von C_i eine Überdeckung für Y_{C_i}. Die so erzeugte Überdeckungen mit den geringsten Kosten ist die optimale, die Bedingungen erfüllende Überdeckung von Y_{C_i}.

Algorithmus 4.2 MinSuperlinearCostCircleCover(Y)

```
Sortiere die festgenagelten Kreise von links nach rechts nach ihrem Mittelpunkt
Cost[i] := 0;
for j = 1 to p + 1 do
  Cost[j] := ∞;
  for i = 1 to j − 1 do
    if C_i und C_j enthalten nicht den Apex des jeweils Anderen ∧
    B(C_i, C_j) ist leer. then
      Cost[j] := min{Cost[j], Cost[i] + f(radius(C_i))};
    end if
  end for
  return Cost[p + 1];
end for
```

Für jeden der $O(n^2)$ festgenagelten Kreise müssen $O(n^2)$ andere festgenagelte Kreise betrachtet werden. Für jedes solche Paar (C_i, C_j) muss außerdem festgestellt werden, ob $B(C_i, C_j)$ leer ist, was bei der trivialen Implementation $O(n)$ Zeit benötigt. Die Gesamtlaufzeit ist also $O(n^5)$.

Jedes $B(C_i, C_j)$ kann in zwei oder drei Abschnitte, die links und rechts von senkrechten Geraden und unten von einem Kreissegment oder der x-Achse beschränkt werden. Nach oben sind die Abschnitte unbeschränkt. Unter Verwendung der Datenstruktur, die in Abschnitt 4.1.1 verwendet wurde um herauszufinden, ob der Bereich oberhalb eines Kreises leer ist, kann getestet werden, ob ein durch ein Kreissegment beschränkter Abschnitt leer ist. Für die durch die x-Achse beschränkten Segmente reicht ein einfacher Blattsuchbaum, in dem die Clients nach der x-Koordinate sortiert sind. In beiden Fällen sind die Kosten des Tests $O(\log n)$. Dadurch wird die Gesamtlaufzeit auf $O(n^4 \log n)$ verbessert.

Theorem 5 *Sei $f : \mathbb{R}_+ \to \mathbb{R}$ eine feste, monoton steigende Kostenfunktion. Dann kann für jede Menge von n Clients und eine beliebige L_p-Metrik eine Überdeckung mit minimalen Gesamtkosten durch x-Achsen-zentrierte Kreise in $O(n^4 \log n)$ berechnet werden.*

Auf einen Beweis wird an dieser Stelle verzichtet. Damit das Lemma für eine beliebige L_p-Metrik gilt, muss allerdings noch eine modifizierte Version des Algorithmus für die L_∞-Metrik angegeben werden.

Da ein L_∞-Kreis ein Quadrat ist, besitzt er keinen eindeutigen Apex. Um dieses Dilemma zu lösen, wird der Apex eines L_∞-Kreises als der Punkt in der rechten, oberen Ecke des Rechtecks definiert. Grundsätzlich wäre es intuitiver, den Punkt auf dem oberen Rand zu wählen, der über dem Zentrum liegt. Dies ist auch möglich. Der rechte, obere Punkt wurde gewählt, da diese Definition zu einer besonders einfachen Form der $B(A, C)$ führt.

Passend zu der Definition des Apex wird P_C für einen L_∞-Kreis C als der Bereich oberhalb und links neben C definiert. Dementsprechend ist jedes $B(A, C)$ die Vereinigung zweier nach oben offener Rechtecke.

Nun ist es zwar möglich, dass ein Quadrat S einer optimalen Überdeckung den Apex eines anderen Quadrats S' enthält. In einem solchen Fall liegt S' links von S und ist kleiner. Konstruiert man nun ein Quadrat S'', indem man S' so weit nach links schiebt, dass dessen rechte Kante auf der linken Kante von $'$ liegt, so überdeckt S'' $S' \setminus S''$ und hat die gleichen Kosten. Folglich existiert zumindest eine optimale Überdeckung, in der kein Quadrat den Apex eines anderen enthält.

Folgende Beobachtung erlaubt uns, den Algorithmus für die L_∞-Metrik weiter zu optimieren: für jedem Teilmenge $Y' \subseteq Y$ existiert eine minimale Überdeckung durch L_∞-Kreise, bei der der am weitesten rechts liegende Client auf dem rechten Rand des am weitesten rechts liegenden Kreises der Überdeckung liegt. Dies liegt daran, dass jeder L_∞-Kreis so weit nach links verschoben werden kann, dass ein Client auf dem rechten Rand liegt, ohne dass ein Client den Kreis verlässt.

Aufgrund der Beobachtung muss jetzt nicht mehr jeder Kreis links von C_i als C_j betrachtet werden. Stattdessen genügt es, nur noch die zu betrachten, auf deren Rand der am weitesten rechts liegende Client aus Y_i liegt. Also kommen nicht mehr $O(n^2)$, sondern nur noch $O(n)$ Kreise in Frage. Die Gesamtlaufzeit verringert sich also um einen linearen Faktor auf $O(n^3 \log n)$

Theorem 6 *Sei $f : \mathbb{R}_+ \to \mathbb{R}$ eine feste, monoton steigende Kostenfunktion. Dann kann für jede Menge von n Clients eine Überdeckung mit minimalen minimaler Gesamtkosten durch x-Achsen-zentrierte Quadraten in $O(n^3 \log n)$ berechnet werden.*

4.2 Approximationsalgorithmen

Als nächstes werden effizientere Algorithmen, die die optimale Lösung mit einem konstanten Faktor approximieren, betrachtet. Für jeden Algorithmus gibt es also eine Konstante c, so dass die berechnete Lösung maximal um den Faktor c schlechter als die optimale Lösung ist. Eine solche Approximation wird c-Approximation genannt. Zunächst wird nur die L_∞-Metrik betrachtet. Am Ende des Unterkapitels wird gezeigt, wie die Ergebnisse auf jede beliebige L_p-Metrik übertragen werden können.

4.2.1 Der Square-Greedy-Cover-Algorithmus (SG)

Der Square-Greedy-Cover-Algorithmus bearbeitet die Clients in absteigender Reihenfolge, sortiert nach dem Abstand von der x-Achse. Für jeden Client p_i, der nicht zuvor als Seiteneffekt des Überdeckens eines anderen Clients überdeckt wurde, wird ein Quadrat platziert, dessen Mittelpunkt auf der x-Achse exakt unter p_i liegt, und dessen Radius der Abstand von p_i

zur x-Achse ist. p_i liegt also exakt mitten auf dem oberen Rand des platzierten Quadrats.

Offensichtlich berechnet der SG-Algorithmus eine Überdeckung der Clients. Bevor jedoch Zeitkomplexität und Approximationsfaktor betrachtet werden, wird das folgende Lemma bewiesen:

Lemma 7 *In einer von SG berechneten Überdeckung wird jeder Punkt in der Ebene (also auch Punkte, an denen kein Client liegt) von maximal zwei Quadraten überdeckt.*

Beweis. Seien S_i und S_j zwei sich überschneidende Quadrate in einer durch SG berechneten Überdeckung und r_i und r_j ihre Radien. Ohne Beschränkung der Allgemeinheit sei S_i vor S_j platziert worden. Sei p_j der Client, für den S_j platziert wurde. Dann kann S_i nicht p_j enthalten, da S_j sonst gar nicht mehr platziert worden wäre. Außerdem ist r_i größer, als der Abstand zwischen p_j und der x-Achse. Folglich ist die einzige Möglichkeit, wie S_i p_j nicht enthalten kann, dass die Zentren von S_i und S_j mehr als r_i von einander entfernt sind. Hieraus folgt, dass S_i das Zentrum von S_j nicht enthalten kann.

Da S_j nach S_i platziert wurde, gilt außerdem $r_j \leq r_i$. Also kann auch S_j nicht das Zentrum von S_i enthalten. Es folgt also, dass für ein beliebiges Paar von Quadraten aus einer von SG berechneten Überdeckung gilt, dass keines das Zentrum des anderen enthält.

Sei p ein beliebiger Punkt aus $S_i \cap S_j$. Sei nun S_k ein Quadrat aus der Überdeckung, das ebenfalls p enthält und I das von $S_i \cap S_j$ überdeckte Intervall der X-Achse. Dann liegt das Zentrum von S_i in I und links von p und das von S_j in I und rechts von p oder umgekehrt. Falls das Zentrum von S_k also außerhalb von I liegt, so überdeckt S_k zwangsläufig eines der Zentren von S_i und S_j. Liegt das Zentrum von S_k jedoch nicht außerhalb von I, so überdeckt $S_i \cap S_j$ das Zentrum von S_k. Folglich enthält entweder $S_i \cap S_j$ das Zentrum von S_k, oder S_k enthält das Zentrum von S_i oder S_j. Beides kann aufgrund der eben getroffenen Aussage nicht zutreffen. Folglich kann unabhängig von der Wahl von p kein solches S_k existieren. Es folgt also, dass in einer von SG berechneten Überdeckung kein drittes Quadrat einen Punkt, der bereits in zwei Quadraten enthalten ist, überdecken kann. □

Unter Zuhilfenahme von Lemma 7 wird nun das folgende Theorem bewiesen:

Theorem 8 *Für eine Menge von n Clients in der Ebene und einer Kostenfunktion $f(r) = r^\alpha$ mit $\alpha \geq 1$ berechnet der Square-Greedy-Cover-Algorithmus eine Überdeckung durch achsenparallele, x-Achsen-zentrierte Quadrate, die maximal die dreifachen Kosten der optimalen Überdeckung hat. Hierfür wird $O(n \log n)$ Zeit benötigt.*

Beweis.

Zunächst wird die Approximationsgüte bewiesen. Der Beweis basiert darauf, jedes Quadrat der von SG berechneten Lösung einem der optimalen Lösung OPT zuzuordnen. Anschließend wird gezeigt, dass die Kosten der Quadrate die einem beliebigen Quadrat aus OPT zugeordnet werden maximal drei mal so groß sind.

Sei $Y = \{p_1, \ldots, p_n\}$ eine Menge von Clientpositionen und S ein Quadrat aus OPT. Seien p_{i_j} die von S überdeckten Clients, für die SG ein Quadrat platziert (die also nicht als Seiteneffekt überdeckt werden). Seien S_{i_j} die zugehörigen von SG berechneten Quadrate. Dann können folgende Aussagen getroffen werden:

- Kein S_{i_j} ist höher, als S. Sonst müsste das zugehörige p_{i_j} mehr als $Radius(S)$ von der x-Achse entfernt sein, was der Annahme, dass S p_{i_j} überdeckt widerspricht.
- Kein S_{i_j} ragt mehr als $Radius(S)$ links oder rechts aus S heraus. Dies liegt daran, dass S alle p_{i_j} und somit auch alle Zentren der S_{i_j} enthält. Desweiteren kann auf jeder Seite maximal ein S_{i_j} aus S herausragen. Wäre dies nicht so, so müsste ein S_{i_j} das Zentrum eines anderen enthalten, was wie im Beweise zu Lemma 7 gezeigt nicht der Fall sein kann.
- Jeder Abschnitt der x-Achse wird von maximal 2 S_{i_j} überdeckt. Dies folgt unmittelbar aus Lemma 7

Hieraus folgt, dass die Summe der Radien der S_{i_j} maximal $\frac{r_S+2\cdot 2\cdot r_S+r_S}{2} = 3 \cdot Radius(S)$ beträgt, wobei r_S der Radius von S ist. Hieraus folgt im Fall $\alpha = 1$ unmittelbar die zu beweisende Approximationsgüte.

Aus $0 < Radius(S_{i_j}) < Radius(S)$ und $\sum_{i_j} Radius(S_{i_j}) \leq Radius(S)$ folgt außerdem $\sum_{i_j} Radius(S_{i_j})^\alpha \leq Radius(S)^\alpha$

Es bleibt also nur noch die Laufzeit zu betrachten. Zu Beginn wird $O(n \log n)$ benötigt, um die Clients nach dem Abstand von der x-Achse zu ordnen. Anschließend werden die Clients einzeln verarbeitet.

Es wird die Verarbeitung des Clients $p_i = (x, y)$ betrachtet. Wurde p_i bereits überdeckt, kann der Algorithmus mit dem nächsten Client fortfahren. Andernfalls wird ein Quadrat S_i mit Mittelpunkt in $(x, 0)$ und Radius y platziert. Da gilt für jeden Client p_j, dessen x-Koordinate im von S_i überdeckten Intervall $[x - y, x + y]$ liegen: Entweder p_j wird von S_i überdeckt, oder p_j wurde zuvor von einem anderen Quadrat überdeckt. Jedes p_j, dass nicht von S_i überdeckt wird, hat mehr als y Abstand von der x-Achse und wird dementsprechend vor p_i verarbeitet.

Alle Clients, die eine x-Koordinate in $[x - y, x + y]$ haben, lassen sich unter Zuhilfenahme eines Blattsuchbaumes mit verketteten Blättern, in dem alle Clients sortiert nach der x-Koordinate enthalten sind, finden. Enthält

dieser Baum zusätzlich nur die Clients, die maximal y Abstand von der x-Achse haben, so können wir die von S als Seiteneffekt überdeckten Clients in $O(\log n + k_i)$ finden, wobei k_i die Anzahl der der als Seiteneffekt überdeckten Clients ist. Diese müssen dann als bereits überdeckt markiert und aus dem Blattsuchbaum entfernt werden. Jeder Client kann in $O(\log n)$ entfernt werden. Also sind die Kosten des Bearbeitens von p_i $O(\log n + k_i \log n)$

Unterm Strich werden also für das Sortieren der Clients und das Initialisieren des Blattsuchbaumes je $O(n \log n)$ Zeit benötigt. Für das Einfangen eines Clients wird $O(\log n + k_i \log n)$ benötigt. Da jeder Client nur einmal aus dem Suchbaum entfernt werden kann, gilt $\sum_{i=1}^{n} k_i = n$. Also wird für das Einfangen der Clients insgesamt $O(n \log n)$ Zeit benötigt und der SG Algorithmus benötigt insgesamt $O(n \log n)$ Zeit. □

4.2.2 Der Square-Greedy-with-Growth-Algorithmus (*SGG*)

In diesem Abschnitt wird ein weiterer Algorithmus vorgestellt, der ebenfalls nur $O(n \log n)$ Zeit benötigt, jedoch einen besseren Approximationsfaktor für lineare Kostenfunktionen ($\alpha = 1$) erreicht. Die Idee des Algorithmus basiert auf der bereits in Abschnitt 4.1.1 gemacht Beobachtung, dass es bei linearen Kosten nie teurer ist, einen Bereich durch einen großen Kreis zu überdecken, als zwei kleinere, überlappende Kreise zu nutzen. Dementsprechend wird der SG Algorithmus so modifiziert, dass ein Quadrat S_i (mit zugehörigem Client p_i) nicht platziert wird, wenn es ein bereits bestehendes Quadrat S_j überschneiden würde. Stattdessen wird S_j so weit vergrößert, dass es auch p_i überdeckt. Dabei wird die p_i abgewandte, senkrechte Kante von S_j nicht bewegt - dass Zentrum von S_j wandert also im Zuge des Vergrößerns auf p_i zu. Sollte S_i zwei Quadrate schneiden, so wird dasjenige vergrößert, dessen Rand näher an p_i liegt. Fälle in denen S_i mehr als zwei Quadrate schneidet, können nicht auftreten, da alle bereits platzierten Quadrate größer sind, als S_i und einander nicht schneiden.

Diese Variante des SG wird Square-Greedy-with-Growth-Algorithmus (*SGG*) genannt.

Theorem 9 *Der SGG berechnet eine Überdeckung von n Clients durch achsenparallele, x-Achsen-zentrierte Quadrate in $O(n \log n)$. Bei linearer Kostenfunktion sind die Kosten dieser Überdeckung maximal doppelt so groß, wie die minimalen Kosten einer Überdeckung durch achsenparallele, x-Achsen-zentrierte Quadrate.*

Beweis. Zunächst werden die Kosten der berechneten Lösung abgeschätzt. Dazu wird jedem Client p_i ein Segment s_i auf der x-Achse zugeordnet. Wurde für p_i ein Quadrat S_i platziert, so wird p_i der von S_i überdeckte Teil zugeordnet. Ein solches s_i sei als Typ 1 bezeichnet. Es hat die Länge des doppelten Abstandes von p_i zur X-Achse und p_i liegt mitten über s_i.

Wurde für p_i ein Quadrat S_j vergrößert, so wird p_i derjenige Teil der x-Achse zugeordnet, der nach dem Vergrößern von S_j überdeckt wird und davor nicht von S_j überdeckt wurde. Da S_j nur auf p_i zu vergrößert wird, ist s_i also ein Intervall auf der x-Achse, dessen Länge dem Abstand von p_i zu S_j vor dem Vergrößern entspricht. Ein solches s_i sei als Typ 2 bezeichnet. Ein s_i vom Typ 2 ist maximal so lang, wie p_i von der X-Achse entfernt ist. Wäre es länger, so wäre p_i nicht durch vergrößern eines Quadrats sondern durch Platzieren überdeckt worden und es wäre ein s_i vom Typ 1 erzeugt wurden.

Offensichtlich entspricht die Summe der Längen der s_i der Länge des von Quadraten überdeckten Teils der x-Achse und somit dem doppelten der Kosten der berechneten Lösung. Folglich reicht es, zu zeigen, dass für jedes Quadrat S aus der optimalen Überdeckung die Summe der Längen der s_i, die zu durch S überdeckten Clients p_i gehören, maximal das doppelte der Breite s von S ergeben.

Kein von S überdeckter Client ist weiter, als $\frac{s}{2}$ von der x-Achse entfernt. Folglich haben die s_i vom Typ 1 maximal die Länge s. Da außerdem die p_i der s_i vom Typ 1 zentral über den s_i und innerhalb von S liegen, ragen sie maximal um $\frac{s}{2}$ auf einer Seite aus S heraus. Die s_i vom Typ 2 sind maximal $\frac{s}{2}$ lang und ragen folglich ebenfalls maximal um $\frac{s}{2}$ aus S heraus. Also müssen die zu S gehörenden s_i sich vollständig auf einem Intervall der Länge $2s$ liegen. Da die s_i außerdem disjunkt sind, haben sie insgesamt eine maximale Länge von $2s$ und der Beweis bezüglich der Approximationsgüte ist geglückt.

Es bleibt also die Zeitkomplexität zu betrachten. Alle Teile des SGG, die wie beim SG ablaufen, haben ebenfalls die gleiche Zeitkomplexität. Zu betrachten bleibt also die Zeit, die benötigt wird, um herauszufinden, ob ein neu platziertes Quadrat ein bereits existierendes schneidet. Zu diesem Zweck wird ein binärer Suchbaum, der die x-Koordinaten der senkrechten Ränder der bereits platzierten Quadrate enthält. Wird ein noch nicht überdeckter Client p_i eingefangen, so werden zunächst die zu p_i benachbarten Kanten der Quadrate im Suchbaum gesucht. Dann wird anhand der Abstände zu diesen Kanten entschieden, ob ein neues Quadrat platziert oder ein bestehendes vergrößert werden muss. Wird ein neues Quadrat platziert, so müssen zwei neue Kanten in den Suchbaum eingefügt werden, wofür $O(\log n)$ Zeit benötigt wird (da der Suchbaum maximal $2n \in O(n)$ enthält). Wird ein bestehendes Quadrat vergrößert, so muss eine Kante im Baum aktualisiert werden, was ebenfalls in $O(\log n)$ Zeit möglich ist. Die zusätzlichen Kosten für das Testen auf Schnitte zwischen potentiell neu platzierten und bestehenden Quadraten betragen also insgesamt $O(n \log n)$ und die Gesamtlaufzeit bleibt $O(n \log n)$. □

Im Gegensatz zu SG bietet SGG keine $O(1)$-Approximation, wenn die überdeckte Fläche als Kostenfunktion dient ($\alpha = 2$). Dies kann anhand des

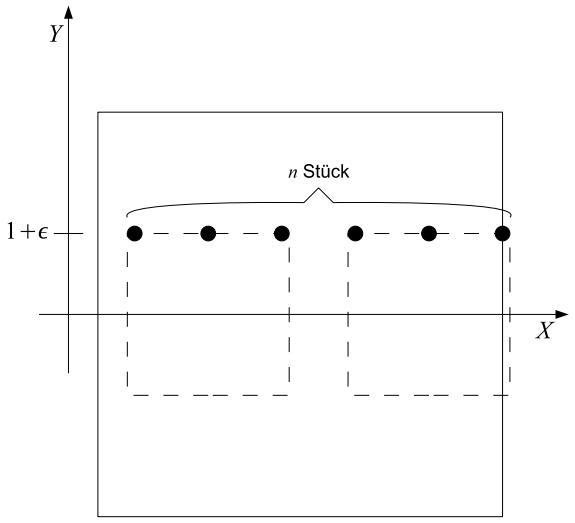


Abbildung 9: Das im Text vorgestellte Beispiel, bei dem die Kosten der SGG-Lösung (durchgezeichnet) quadratisch wachsen, die optimalen Kosten (gestrichelt) jedoch nur linear (hier für $n = 6$).

folgenden Beispiels verdeutlicht werden:

n Clients werden auf der Höhe $1 + \epsilon$ platziert. Die Abstände zwischen den Clients betragen 1. Da alle Clients gleich weit von der x-Achse entfernt sind, gibt es nun keine vorgegebene Reihenfolge, in der die Clients eingefangen werden. Es sei angenommen, dass der SGG die Clients von links nach rechts einfängt. Zunächst wird ein Quadrat mit Seitenlänge $2 + 2\epsilon$, dessen Zentrum unter dem linken Quadrat liegt, platziert. Der zweite Client wird von diesem Quadrat gleich mit eingefangen. Für den dritten Client müsste ein weiteres Quadrat mit Seitenlänge $2 + 2\epsilon$ platziert werden. Da der Dritte Client jedoch nur $1 - \epsilon$ weit von dem rechten Rand des ersten Quadrates entfernt liegt, würde das neu platzierte Quadrat dieses schneiden. Also wird das bereits bestehende Quadrat um $1-\epsilon$ vergrößert. Nun ist der rechte Rand des Quadrates 1 weit von dem vierten Client entfernt. Da dieser ebenfalls ein Quadrat der Seitenlänge $2+2\epsilon$ benötigen würde, wird auch der vierte Client von dem bereits bestehenden Quadrat eingefangen. Analog werden alle weiteren Clients durch Vergrößern des Quadrates eingefangen. Also überdeckt SGG die Clients mit einem großen Quadrat, das $1-\epsilon$ links des ersten Clients beginnt und bis zu letzten Client geht. Es hat also einen Durchmesser von $n + 1 + \epsilon$ und eine Fläche von $(n + 1 + \epsilon)^2$.

Optimal wäre es, immer drei Clients von einem Quadrat der Seitenlänge $2 + 2\epsilon$ zu überdecken. Diese Lösung würde aus $\frac{n}{3}$ Qadraten der Seitenlänge $2 + 2\epsilon$ bestehen und würde eine Fläche von $\frac{n}{3} \cdot (2 + 2\epsilon)^2$ benötigen. Das Verhältnis zwischen der vom SGG berechneten und der optimalen Fläche

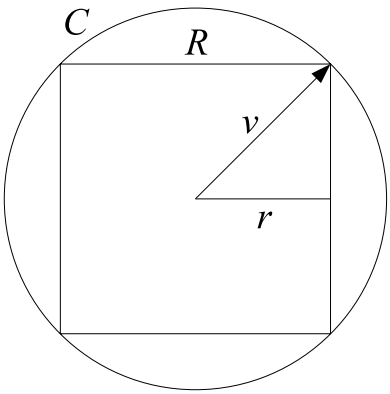


Abbildung 10: Übertragung auf beliebige L_p-Metriken, in diesem Fall die L_2-Metrik

ist also $\frac{(n+1+\epsilon)^2}{\frac{2(1+\epsilon)n}{3}} \geq \frac{3n^2}{2(1+\epsilon)n} = \frac{3n}{1+\epsilon} \geq \frac{3}{4}n \in \Omega(n)$. Die von SGG berechnete Lösung hätte also die $\Omega(n)$-fachen Kosten der optimalen Lösung.

4.2.3 Übertragung der Lösungen auf andere L_p-Metriken

Sowohl SG als auch SGG arbeiten mit der L_∞-Metrik. Will man Sie für andere Metriken verwenden, so kann man sich mit folgendem Trick behelfen: Zunächst werden die Clients durch Quadrate überdeckt. Anschließend wird um jedes Quadrat ein Kreis der zu verwendenden L_p-Metrik gelegt. Dies wird in Abbildung 10 dargestellt: Der L_p-Metrik-Kreis C wird so um das Quadrat R gelegt, dass er es in den Ecken berührt und folglich R komplett überdeckt. Der Radius von C ist die Länge des Vektors $v = (r, r)^T$ in der L_p-Metrik. Er beträgt also $\sqrt[p]{r^p + r^p} = \sqrt[p]{2r^p} = \sqrt[p]{2}r$, also das $\sqrt[p]{2}$-fache des Radius von R. Die Kosten L_p-Metrik-Überdeckung betragen also das $r^{\frac{\alpha}{p}}$-fache der Kosten der Überdeckung in L_∞-Metrik. Die Kosten der optimalen Überdeckung mit L_p-Metrik-Kreisen sind mindestens so groß, wie die der L_∞-Metrik-Überdeckung, da man aus jeder L_p-Metrik-Überdeckung eine L_∞-Metrik-Überdeckung gleicher Kosten machen kann, indem man die L_p-Metrik-Kreise durch Quadrate mit gleichem Radius ersetzt. Es folgt Theorem 10.

Theorem 10 *Für eine Menge von n Clients in der Ebene und eine Kostenfunktion $f(r) = r^\alpha$ mit $\alpha \geq 1$ berechnet der Square-Greedy-Cover-Algorithmus eine Überdeckung durch x-Achsen-zentrierte L_p-Metrik-Kreise, die maximal die $3 \cdot 2^{\frac{\alpha}{p}}$-fachen Kosten der optimalen Überdeckung hat. Für $\alpha = 1$ berechnet der Square-Greedy-with-Growth-Algorithmus eine Überdeckung durch L_p-Metrik-Kreise, die maximal das $2 \cdot 2^{\frac{1}{p}}$-fache der optimalen Kosten hat.*

5 Finden der besten achsenparallelen Gerade

In diesem Abschnitt wird das folgende Problem betrachtet: Gegeben ist eine Menge von Clients $Y = \{p_1, \ldots, p_n\}$. Gesucht wird eine horizontale Gerade g und eine Überdeckung durch g-zentrierte Kreise, so dass zu einer festen Kostenfunktion $f(r) = r^\alpha$ die Summe der Kosten der Kreise minimiert wird. Im Gegensatz zu Abschnitt 4 kann die horizontale Linie, auf der die Kreismittelpunkte liegen, also frei gewählt werden.

5.1 Unberechenbarkeit durch Wurzeln

In diesem Abschnitt wird gezeigt, dass das Problem selbst für $\alpha = 1$ unberechenbar durch Wurzeln ist. Der Beweis erfolgt ähnlich wie das in [6, 7] dargestellte Verfahren.

Theorem 11 *Seien $c(t) = \sum_i r_i$ die minimalen Kosten einer Überdeckung durch Kreise, deren Zentren auf einer Gerade mit Geradengleichung $y = t$ liegen. Es existiert eine Menge Y, so dass der Wert t_0, der $c(t)$ minimiert unberechenbar durch Wurzeln ist.*

Beweis. Der Beweis konstruiert eine solche Menge Y und zeigt durch Ableiten von $c(t)$, dass t_0 Nullstelle eines Polynoms ist, dass erwiesenermaßen unlösbar durch Wurzeln ist.

Zunächst werden jedoch folgende Definitionen und Fakten, die in der Literatur, beispielsweise in [22] nachgelesen werden können, aufgelistet: Ein Polynom $f(x) : \mathbb{Q} \to \mathbb{Q}$ ist lösbar durch Wurzeln, wenn seine Nullstellen durch rationale Zahlen, die Körper-Operationen ($+$, $\cdot$, $-x$, x^{-1}) und das Ziehen k-ter Wurzeln dargestellt werden können. Der Zerfällungskörper von $f(x)$ ist die kleinste Untermenge der komplexen Zahlen, die alle Nullstellen von $f(x)$ und alle rationalen Zahlen enthält und die den Körperaxiomen gehorcht. (Insbesondere müssen also alle inversen Elemente enthalten sein). Sei F der Zerfällungskörper von $f(x)$. Dann ist die Galoisgruppe von $f(x)$ über $\mathbb{Q}$ die Menge aller bijektiven, linearen Abbildungen $F \to F$, die alle Elemente aus $\mathbb{Q}$ auf sich selbst abbilden. Falls die Galoisgruppe von $f(x)$ über $\mathbb{Q}$ eine symmetrische Gruppe über fünf oder mehr Elementen ist (also eine Gruppe, die aus allen Permutationen einer n-elementigen Menge mit $n \geq 5$ besteht), so ist $f(x)$ nicht berechenbar durch Wurzeln.

Basierend auf diesen Definitionen und Aussagen wird nun gezeigt, dass die beste horizontale Gerade g zur Überdeckung der Client-Menge $Y = \{(3, 4), (-3, -2), (102, 2), /98, -2), (200, 2)\}$ unberechenbar durch Wurzeln ist. Offensichtlich ist es niemals optimal, die Gerade oberhalb des oberen oder unterhalb des unteren Clients zu platzieren. Folglich muss t_0 nur im Bereich $t \in [-2, 2]$ gesucht werden. Durch Fallunterscheidung kann außerdem gezeigt werden, dass es immer optimal ist, einen Kreis um die ersten beiden Clients, einen um den dritten und den vierten Client und

einen um den fünften zu platzieren. Hieraus ergibt sich die Kostenfunktion $c(t) = \sqrt{2(t-1)^2+18} + \sqrt{2t^2+8} + (2-t)$. Um das Minimum zu finden wird die Ableitung auf 0 gesetzt: $c'(t) = \frac{2(t-1)}{\sqrt{2(t-1)^2+18}} + \frac{2t}{\sqrt{2t^2+8}} - 1 = 0$ Außerdem gilt $c''(t) > 0$ für alle t. Hieraus ergibt sich das Minimum $c(y) \approx 8,3327196$ bei $t_0 \approx 1.4024709$. Dies wiederum ist Nullstelle des Polynoms $f(t) = 1024 + 512t - 1600t^2 + 1536t^3 - 960t^4 + 368t^5 - 172t^6 + 28t^7 - 7t^8$. Unter Zuhilfenahme des Computer-Algebra-Systems GAP [23] wurde berechnet, dass die Galoisgruppe von $f(t)$ die symmetrische Gruppe S_8 (über eine 8-elementige Menge) ist [2]. Folglich ist $f(t)$ unberechenbar durch Wurzeln.

□

5.2 Ein FPTAS zum Finden der besten Lösung

5.2.1 Lineare Kosten

Nun wird ein FPTAS (Fully polynomial-Time Approximation Scheme) zur Approximation der besten Lösung gefunden werden. Ein FPTAS ist ein Algorithmus, der für jedes $\epsilon > 0$ eine Lösung findet, die maximal das $1+\epsilon$-fache der minimalen Kosten hat. Dabei hängt die Laufzeit des Algorithmus nur polynomiell von der Größe der Eingabe und $\frac{1}{\epsilon}$ ab.

Zunächst wird mit $\alpha = 1$ begonnen. Sei d der Abstand zwischen dem höchsten und dem tiefsten Client und OPT die Kosten der optimalen Überdeckung. Offensichtlich gilt $\frac{d}{2} \leq OPT \leq n\frac{d}{2}$ (Können alle Clients durch einen Kreis abgedeckt werden, so muss dieser einen Radius von mindestens $\frac{d}{2}$ habe; wird die Linie in die Mitte zwischen den höchsten und den niedrigsten Client gelegt, so kann jeder Client mit einem eigenen Kreis mit Radius von maximal $\frac{d}{2}$ abgedeckt werden). Nun wird er horizontale Streifen der Höhe d, der alle Clients enthält, in $\frac{2n}{\epsilon}$ Streifen der Höhe $\delta = \frac{d\epsilon}{2n}$ unterteilt. Dazu werden $\frac{2n}{\epsilon} - 1$ parallele Geraden g_i mit Abstand δ verwendet. Für jede dieser Geraden wird der Dynamisches-Programmieren-Algorithmus aus Abschnitt 4.1 zum Finden der besten Überdeckung verwendet. Die beste dieser Lösungen ist dann die approximative Lösung des Problems. Im Folgenden wird der Fehler dieser Lösung abgeschätzt.

Sei g^* die optimale Gerade. g^* kann auf die nächste Gerade g_i verschoben werden, wenn der Radius jedes Kreises um bis zu δ erhöht wird. Die Gesamtkosten erhöhen sich dabei um maximal $n \cdot \delta = \epsilon \cdot \frac{d}{2} \leq \epsilon \cdot OPT$. Die Kosten erhöhen sich also maximal um den Faktor $1+\epsilon$. Anders ausgedrückt, gibt es ein g_i dessen optimale Überdeckung maximal $(1+\epsilon)OPT$ Kosten hat. Dieses g_i wird von dem FPTAS auf jeden Fall gefunden (es kann allerdings auch ein anderes g_i gefunden werden, dessen Kosten noch niedriger sind, als die der auf das nächste g_i verschobene optimalen Lösung).

Für das Berechnen der optimalen Lösung auf einer der Geraden g_i werden $O(n^2 \log n)$ Zeit benötigt (siehe Theorem 4). Der Algorithmus muss $\frac{2n}{\epsilon} -$

$1 = O(\frac{n}{\epsilon})$ solcher Lösungen berechnen. Die Zeitkomplexität beträgt also $O(\frac{n^3}{\epsilon} \log n)$.

5.2.2 Superlineare Kosten

Nun werden die Ergebnisse aus Abschnitt 5.2.1 auf den superlinearen Fall ($\alpha > 1$) verallgemeinert. Nun sei $PSEUDO\text{-}OPT$ die beste Lösung, die auf einem g_i liegt, also die beste Lösung, die von dem FPTAS gefunden wird. $SHIFT$ sei die Lösung, die man erhält, indem man die optimale Lösung auf das nächste g_i verschiebt. Dann kann $PSEUDO\text{-}OPT = SHIFT$ sein - dies ist jedoch nicht zwangsläufig der Fall. Seien m die Anzahl der Kreise in der optimalen Lösung und seien $r_1, \ldots, r_m$ die Radien der Kreise in OPT. Desweiteren sei $\alpha > 1$. Dann gilt:

- $PSEUDO\text{-}OPT \leq SHIFT \leq \sum_{i=1}^{m}(r_i + \delta)^\alpha$

($SHIFT$ wird auf jeden Fall gefunden und jeder Kreis in OPT muss um maximal δ erhöht werden, wenn die optimale Lösung auf das nächste g_i verschoben wird) Allgemein gilt $(x+\delta)^\alpha \leq x^\alpha + \delta\alpha(x+\delta)^\alpha$ für $x \geq 0$, $\delta > 0$ und $\alpha \geq 1$. Dies lässt sich wie folgt verdeutlichen: $(x+\delta)^\alpha$ ist der Wert Funktion $h(x) = x^\alpha$ an der Stelle $x+\delta$. $\alpha(x+\delta)^\alpha$ ist die Ableitung $h'(x)$ an der Stelle $x+\delta$. Exakt berechnet gilt $h(x+\delta) = h(x) + \int_{t=x}^{x+\delta} h'(t)dt$. Aus $\alpha > 1$ folgt, dass $h'(x)$ monoton steigt. Daraus wiederum folgt, dass $\int_{t=x}^{x+\delta} h'(t)dt \leq \int_{t=x}^{x+\delta} h'(x+\delta)dt = \delta h'(x+\delta)$. Also gilt $h(x+\delta) \leq h(x) + \delta h'(x+\delta)$ und die Aussage ist bewiesen. Setzen wir sie in unsere Abschätzung ein, so folgt:

- $\sum_{i=1}^{m}(r_i+\delta)^\alpha \leq \sum_{i=1}^{m} r_i^\alpha + \delta\alpha \sum_{i=1}^{m}(r_i+\delta)^{\alpha-1}$

Aus $\delta \leq d$, $r_i \leq d$, $(\frac{d}{2})^\alpha \leq OPT$ und $m \leq n$ (mehr als ein Kreis pro Client kann nicht optimal sein) folgt dann:

$$\begin{aligned}
\sum_{i=1}^{m} r_i^\alpha + \delta\alpha \sum_{i=1}^{m}(r_i+\delta)^{\alpha-1} &\leq OPT + \delta\alpha \sum_{i=1}^{m}(d+d)^{\alpha-1} \\
&= OPT + \delta\alpha m 2^{\alpha-1} d^\alpha \frac{1}{d} \\
&= OPT + \delta\alpha m 2^{2\alpha-1} (\frac{d}{2})^\alpha \frac{1}{d} \\
&\leq OPT + \delta\alpha n 2^{2\alpha-1} OPT \frac{1}{d} \\
&= OPT(1 + \frac{\delta\alpha n 2^{2\alpha-1}}{d})
\end{aligned}$$

Also folgt $PSEUDO\text{-}OPT \leq OPT(1 + \frac{\delta\alpha n 2^{2\alpha-1}}{d})$. Das gewünschte Ergebnis $PSEUDO\text{-}OPT \leq (1+\epsilon)OPT$ kann nun erreicht werden, indem $\delta = \frac{\epsilon}{\alpha 2^{2\alpha-1} n}$ gewählt wird.

Wegen $\alpha > 1$ benötigen wir nun gemäß Theorem 5 $O(n^4 \log n)$ Zeit für das Finden einer optimalen Lösung. Es werden $O(d/\delta) = O(n/\epsilon)$ Geraden g_i betrachtet. Also benötigt das FPTAS für $\alpha > 1$ $O(\frac{n^5}{\epsilon} \log n)$ Zeit.

Die Ergebnisse von Abschnitt 5.2.1 und Abschnitt 5.2.2 können wie folgt zusammengefasst werden:

Theorem 12 *Seien n Clients in der Ebene und ein festes $\alpha \geq 1$ gegeben. Dann existiert ein FPTAS, dass die optimale Gerade g und g-zentrierte Kreise findet, die die Clients überdecken und minimale Gesamtkosten haben. Im linearen Fall ($\alpha = 1$) benötigt das FPTAS $O(\frac{n^3}{eps} \log n)$ und im superlinearen ($\alpha > 1$) $O(\frac{n^5}{eps} \log n)$ Zeit.*

6 Approximation der besten Gerade mit beliebiger Ausrichtung

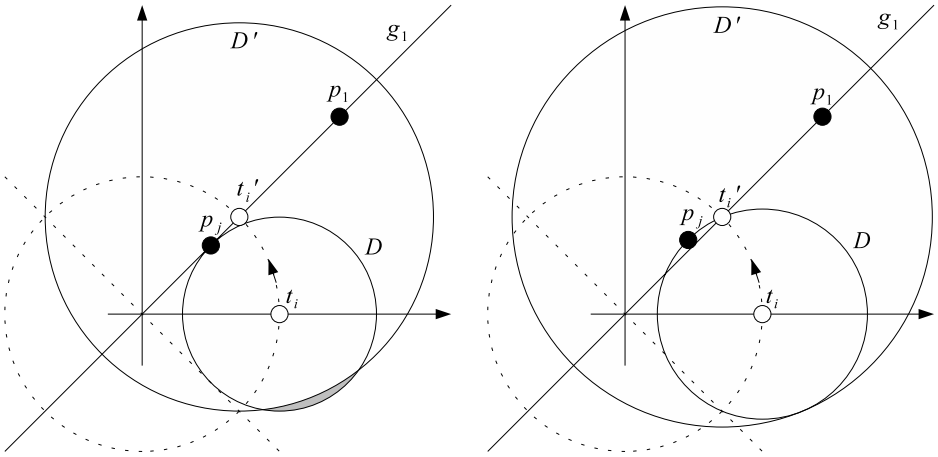
In diesem Abschnitt wird das Problem aus Abschnitt 5 dahingehend modifiziert, dass die Gerade g beliebige Orientierung haben darf, also nicht zwangsläufig achsenparallel sein muss. Weiterhin wird von einer linearen Kostenfunktion ($\alpha = 1$) ausgegangen. Es wird eine simple Approximation angegeben, die eine Lösung berechnet, deren Kosten maximal um einen konstanten Faktor größer sind, als die einer optimalen Lösung. Sie basieren auf folgender Beobachtung:

Lemma 13 *Sei $k \geq 1$, g eine Gerade und $\mathcal{D}$ eine Menge von k g-zentrierten Kreisen, die die Clientmenge Y überdecken. Sei p_0 ein beliebiger Punkt auf g. Dann existieren eine Gerade g', eine Menge $\mathcal{D}'$ von k g'-zentrierten Kreisen, die Y überdecken und maximal die 2^α-fachen Kosten von $\mathcal{D}$ haben.*

Beweis.

Ohne Beschränkung der Allgemeinheit sei g die x-Achse, p_0 sei der Ursprung und außerhalb von p_0 liege kein Client auf g. Die ersten beiden Bedingungen lassen sich durch Rotation und Verschieben aller Clients erreichen. Die letzte Bedingung lässt sich durch eine kleine Perturbation herbeiführen. Sei $Y = \{p_1, \ldots, p_n\}$ und $p_i = (x_i, y_i)$. Desweiteren sei g_i die Gerade, die durch $(0, 0)$ und p_i läuft und m_i die zugehörige Steigung: $m_i = \frac{y_i}{x_i}$. Desweiteren seien die Clients in Y aufsteigend sortiert nach dem Betrag der Steigungen der zugehörigen Geraden: $|m_1| \leq \ldots \leq |m_n|$. Hier wird nur der Fall, in dem p_1 im ersten Sektor der Koordinatensystems liegt, also $x_1 > 0$ und $y_1 \geq 0$ gilt, betrachtet. Andere Fälle können jedoch analog betrachtet werden.

Nun wird für jeden Kreis $D_i = (t_i, r_i)$ aus $\mathcal{D}$ ein Kreis $D'_i = (t'_i, r'_i)$ in $\mathcal{D}'$ wie folgt konstruiert: t_i wird um den Koordinatenursprung um $-\arctan(m_1)$ (also auf g_1) gedreht. r'_i wird auf das doppelte von r_i gesetzt. Folglich ist $\mathcal{D}$



(a) Der Abstand zwischen t_i und t_i' ist größer als r_i. In der Folge ist $D \setminus D'$ nicht leer.

(b) Bereits bei in geringem Maße größeren r_i (grau unterlegt) verschwindet $D \setminus D'$ ganz.

Abbildung 11: $D \setminus D'$

g_1-zentriert und hat die 2^α-fachen Kosten. Es bleibt also zu zeigen, dass $\mathcal{D}'$ auch tatsächlich Y überdeckt.

Folgende Beobachtung hilft bei der Argumentation: Sei D_i ein Kreis aus $\mathcal{D}$ und p_j ein Client, der von D_i überdeckt wird. Dann schneidet oder berührt D_i g_j. Da g_1 die Gerade mit der geringsten Steigung ist, folgt hieraus, dass D_i auch g_1 schneidet oder berührt. Folglich ist r_i mindestens so groß, wie der Abstand zwischen t_i und g_1. Allerdings verschiebt eine Rotation um den Ursprung t_i nicht auf den nächstgelegenen Punkt auf g_1. Darum kann der Abstand zwischen t_i und t_i' größer als r_i sein, wenn r_i nur geringfügig größer ist, als der Abstand von t_i zu g_1. Diese Situation wird in Abbildung 11(a) dargestellt: Der grau unterlegte Bereich liegt in D und wird nicht von D' überdeckt. Ist r_i nur ein wenig größer, so verschwindet der grau unterlegte Bereich komplett und D' überdeckt D komplett. Diese Situation wird in Abbildung 11(b) dargestellt. Desweiteren ist ersichtlich, dass der graue Bereich immer komplett oberhalb der an der x-Achse gespiegelten Gerade g_1 liegt. Läge in dem grauen Bereich also ein Client p_k, so wäre $|m_k|$ kleiner, als $|m_1|$. Dies widerspricht jedoch der Wahl von p_1. Es folgt also, dass $D \setminus D'$ entweder die leere Menge ist, oder dass in $D \setminus D'$ kein Client liegt. Folglich überdeckt D' alle Clients aus Y und der Beweis ist geglückt.

□

Wendet man Lemma 13 zwei mal an, so ergibt sich, dass eine Gerade $g_{i,j}$, die durch die Clients $p_i, p_j \in Y$ läuft, und eine zugehörige, $g_{i,j}$-zentrierte Überdeckung existieren, so dass die Kosten der Überdeckung höchstens das $2^{2\alpha}$-fache der minimalen Kosten beträgt. Folglich lässt das Problem sich ap-

proximativ lösen, indem man einen Algorithmus zur approximativen oder exakten Lösung des Problems bei gegebener fester Gerade auf $O(n^2)$ verschiedene Geraden anwendet, und das beste Ergebnis behält. Folgend aus Theorem 4 erhält man also:

Theorem 14 *Seien Y, eine Menge von n Clients in der Ebene, und ein festes $\alpha \geq 1$ gegeben. Dann lässt sich in $O(n^4 \log n)$ Zeit eine Menge von Kreisen, deren Zentren auf einer Geraden liegen, mit Gesamtkosten, die maximal das 4^α-fache der minimalen Kosten betragen, finden.*

Literatur

[1] P. Agarwal and M. Sharir. Efficient algorithms for geometric optimization. *ACM Computing Surveys (CSUR)*, 30(4):412–458, 1998.

[2] H. Alt, E. Arkin, H. Brönnimann, J. Erickson, S. Fekete, C. Knauer, J. Lenchner, J. Mitchell, and K. Whittlesey. Minimum-cost coverage of point sets by disks. *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 449–458, 2006.

[3] E. Arkin, S. Fekete, and J. Mitchell. Approximation algorithms for lawn mowing and milling. *Computational Geometry*, 17(1-2):25–50, 2000.

[4] E. Arkin and R. Hassin. Approximation algorithms for the geometric covering salesman problem. *Discrete Applied Mathematics*, 55(3):197–218, 1994.

[5] S. Arora, P. Raghavan, and S. Rao. *Approximation schemes for Euclidean k-medians and related problems.* ACM Press New York, NY, USA, 1998.

[6] C. Bajaj. Proving geometric algorithm non-solvability: an application of factoring polynomials. *Journal of Symbolic Computation*, 2(1):99–102, 1986.

[7] C. Bajaj. The algebraic degree of geometric optimization problems. *Discrete and Computational Geometry*, 3(1):177–191, 1988.

[8] V. Bilo, I. Caragiannis, C. Kaklamanis, and P. Kanellopoulos. Geometric clustering to minimize the sum of cluster sizes. *Proc. 13th European Symp. Algorithms*, 2005.

[9] H. Brönnimann and M. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete and Computational Geometry*, 14(1):463–479, 1995.

[10] A. Clementi, P. Penna, and R. Silvestri. On the Power Assignment Problem in Radio Networks, 2000.

[11] A. E. F. Clementi, P. Penna, and R. Silvestri. On the Power Assignment Problem in Radio Networks. Technical Report TR00-054, Electronic Colloquium on Computational Complexity, 2000.

[12] T. Cormen, C. Leiserson, R. Rivest, and S. C. *Introduction to Algorithms*. Cambridge: The MIT Press, 2001.

[13] A. Dumitrescu and J. S. B. Mitchell. Approximation algorithms for tsp with neighborhoods in the plane. *J. Algorithms*, 48(1):135–159, 2003.

[14] T. Erlebach, K. Jansen, and E. Seidel. Polynomial-time approximation scheme for geometric graphs. *SIAM J. Computing*, 34(6):1302–1323, 2005.

[15] S. Fekete, R. Klein, and A. Nuechter. Searching with an autonomous robot. *Proceedings of the twentieth annual symposium on Computational geometry*, pages 449–450, 2004.

[16] S. Fekete, R. Klein, and A. Nuechter. Online Searching with an Autonomous Robot. *Algorithmic Foundations of Robotics VI*, 17:139–154, 2005.

[17] T. GONZALEZ. Covering a set of points in multidimensional space. *Information processing letters*, 40(4):181–188, 1991.

[18] J. Hershberger. Minimizing the sum of diameters efficiently. *Computational Geometry: Theory and Applications*, 2(2):111–118, 1992.

[19] D. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM (JACM)*, 32(1):130–136, 1985.

[20] N. Lev-Tov and D. Peleg. Polynomial time approximation schemes for base station coverage with minimum total radii. *Computer Networks*, 47(4):489–501, 2005.

[21] K. Pahlavan and A. Levesque. *Wireless information networks*. Wiley-Interscience New York, NY, USA, 2005.

[22] J. Rotman. *Advanced modern algebra*. Prentice Hall Upper Saddle River, NJ, 2002.

[23] The GAP Group. GAP – groups, algorithms, and programming Version 4.4, 2005.