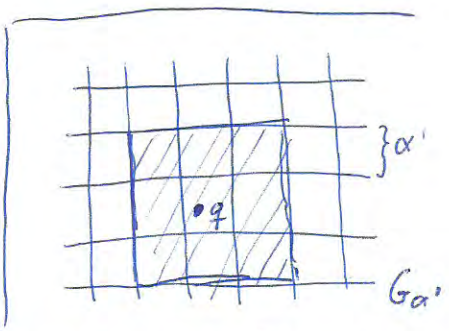
Aufgabe 1.2. im Buch (Cluster Radius berechnen)

(A) Geben Sie einen $O(n + k \log n)$-Zeit Algorithmus, der eine Zahl α ausgibt mit $r \leq \alpha \leq 10r$

Wir lösen zunächst das Entscheidungsproblem approximativ:

(A') Geben Sie einen $O(n+k)$-Zeit Algorithmus, der für ein gegebenes α' ausgibt:

$$\begin{cases} \text{wahr, falls } r \leq \alpha' \\ \text{falsch, falls } r > \alpha' \cdot 2\sqrt{2} \\ \text{wahr oder falsch, sonst} \end{cases}$$



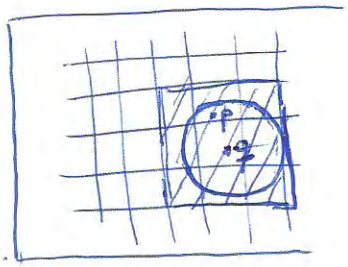
Algorithmus 1:

1. Für jedes $q \in C$: Markiere die Zellen von $G_{\alpha'}$ des Gitterclusters von q. D.h., berechne $id(q)$ und speichere diesen Index und die der 8 angrenzenden Zellen in einer Hash-Tabelle.
2. Für jedes $p \in P$: Prüfe, ob $id(p)$ in der Hashtabelle ist. Falls nicht, gib "falsch" zurück. Falls für alle $p \in P$ $id(p)$ in der Tabelle ist, gib "wahr" zurück

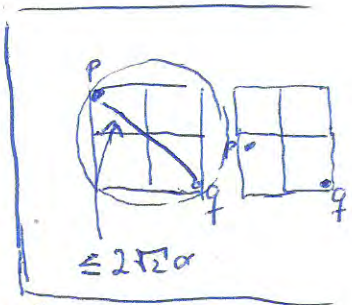
Laufzeit: Schritt 1: $O(k)$, Schritt 2: $O(n)$, also $O(n+k)$ insgesamt.

Korrektheit.

1. Fall: $r \leq \alpha'$. D.h. $\forall p \in P \; \exists q \in C \; \|p-q\| \leq \alpha'$. Hieraus folgt das p in der Zelle von q oder in einer benachbarten Zelle von q in $G_{\alpha'}$ liegt. Diese Zelle wurde in Schritt 1 markiert und in Schritt 2 gefunden. Da dies für jedes P gilt, gibt der Algorithmus - wie verlangt - "wahr" zurück



2. Fall: $r > 2\sqrt{2}\alpha'$: D.h. $\exists p \in P \; \forall q \in C \; \|p-q\| > 2\sqrt{2}\alpha'$. Angenommen die Zelle von p sei markiert. Dann läge in dieser Zelle oder einer benachbarten ein $q \in C$. Der Durchmesser von 2 benachbarten Zellen ist maximal $2\sqrt{2}\alpha'$. Dann wäre aber auch $\|p-q\| \leq 2\sqrt{2}\alpha'$, ein Widerspruch. Also ist die Zelle von p nicht markiert und der Algorithmus gibt - wie verlangt - "falsch" zurück



3. Fall: $\alpha' < r \leq 2\sqrt{2}\alpha'$. Der Algorithmus kann wahr oder falsch zurückgeben, ist also in diesem Fall immer korrekt.

Um einen Algorithmus für A zu entwickeln, verwenden wir Algorithmus 1 innerhalb eines randomisierten Algorithmus, in dem wir gegebenfalls α erhöhen. Statt beliebiger α verwenden wir nur 2er-Potenzen als Seitenlängen. Dazu definieren wir:

$r(p) = \min_{q \in C} \|p-q\|$

$\alpha(p)$ = kleinstes 2^i $(i \in \mathbb{Z})$, so dass ein $q \in C$ im Gridcluster von p in $G_{\alpha(p)}$ ist.

Es gilt: $\frac{r(p)}{2\sqrt{2}} \leq \alpha(p) \leq 2\, r(p)$

Beweis: Für q im Gridcluster von p gilt $\|p-q\| \leq 2\sqrt{2}\,\alpha(p)$ und somit $\frac{r(p)}{2\sqrt{2}} \leq \alpha(p)$. Da kein $q \in C$ im Gridcluster von p für $G_{\frac{\alpha(p)}{2}}$ liegt, gilt für alle q: $\|p-q\| \geq \frac{\alpha(p)}{2}$. □

Algorithmus 2:

(1) Berechne eine zufällige Permutation der Punkte in P, $\langle p_1, \dots, p_n \rangle$

(2) $\alpha = \alpha(p_1)$

(3) Führe Schritt 1 von Alg. 1 für das Gitter G_α aus

(4) For $i = 2, \dots, n$:
Prüfe ob die Zelle von p_i markiert ist.
Falls nicht, setze $\alpha = \alpha(p_i)$ und führe Schritt 1 von Alg 1 für das neue Gitter G_α aus

(5) return $2\sqrt{2}\,\alpha$

Korrektheit:

Mit α_i bezeichnen wir das α nach Iteration i. Es gilt $\alpha_i \leq \alpha_{i+1}$, da, falls α sich ändert, ~~xxx~~ p kein q im Gridcluster hatte und α vergrößert wird.

(1) In Schritt 2-4 wird α so gewählt, dass p_i ein q im Gridcluster hat. Da wir nur 2er Potenzen als α haben, gilt dies auch, wenn wir α erhöhen. Daraus folgt: $r(p) \leq 2\sqrt{2}\,\alpha_n$. Da wir $2\sqrt{2}\alpha_n$ zurückgeben, gilt $\max_{p \in P} r(p) \leq \alpha$.

(2) $\alpha_n = \alpha(p_i)$ für ein $1 \leq i \leq n$ und damit $\alpha_n \leq 2\, r(p)$.
Daraus folgt $\alpha = 2\sqrt{2}\,\alpha_n \leq 4\sqrt{2}\, r(p) \leq 10 \max_{p \in P} r(p)$

Aus (1) + (2) folgt die Korrektheit.

Laufzeit:

Schritt 1: $O(n)$

Schritt 2: In $O(k)$ Zeit berechnen wir $r(p_i)$. In $O(1)$ Zeit können wir dann $2^{\lceil \log r(p_i) \rceil}$ berechnen.

Da $\frac{r(p)}{2\sqrt{2}} \leq \alpha(p) \leq r(p)$, kommen für $\alpha(p)$ nur 3 Werte in Frage, die wir in $O(k)$ Zeit testen können

Schritt 3: $O(n)$

Schritt 5: $O(1)$

Schritt 4: Wir müssen das Gitter nur neu berechnen, wenn $\alpha(p_i) > \max_{1 \leq j < i} \alpha(p_j)$. Sei $P_i = \{p_1, \dots, p_i\}$.

Wenn es kein $p \in P_i$ gibt mit $\alpha(p) > \max_{p' \in P \setminus \{p\}} \alpha(p')$, dann muss das Gitter auch nicht neu berechnet werden.

Wenn es ein solches p gibt, dann ist die Wahrscheinlichkeit, dass $p = p_i$: $1/i$. Wenn das Gitter neu berechnet wird, ist der Zeitaufwand $O(k)$

Daher ist der erwartete Aufwand für Schritt 4.

$$\sum_{i=2}^{n} O(1) + \frac{O(k)}{i} \leq O(n) + O(k) \cdot \sum_{i=2}^{n} \frac{1}{i} = O(n + k \log n).$$

Es ergibt sich eine Gesamtlaufzeit von $O(n + k \log n)$.

Anmerkung: Es wäre einfacher gewesen, $\alpha(p) = r(p)$ zu wählen. Die Korrektheit kann dann auf die gleiche Art bewiesen werden (sogar mit einem Faktor 2 weniger). Das Problem ist dann aber in der Laufzeitanalyse von Schritt 4.

Das Problem ist, dass dann α_i von der Reihenfolge von $p_1, \dots, p_i$ abhängt. Dadurch gibt es nicht unbedingt nur ein $p \in \{p_1, \dots, p_i\}$, dass zu einer Änderung des Gitters führt.

(B) Geben Sie einen $O(n + \frac{k}{\varepsilon^2} \log n)$-Zeit Algorithmus, der eine Zahl α ausgibt mit $r \le \alpha \le (1+\varepsilon) r$

Wir können annehmen, dass $\varepsilon \le 1$

Algorithmus:

(1) Berechne ein α' mit $r \le \alpha' \le 10 r$

(2) $\beta := \frac{\alpha' \cdot \varepsilon}{20 \cdot \sqrt{2}}$

(3) Berechne zufällige Permutation von P : $\langle p_1, \dots, p_n \rangle$

(4) Markiere alle Zellen in G_β, die von den $B_{r(p_1)}(q)$ geschnitten werden, d.h. von den Kreisscheiben mit Radius $r(p_1)$, um alle $q \in C$. Setze $\alpha'' = r(p_1)$.

(5) For $i = 2, \dots, n$:
Falls Zelle von p_i nicht markiert, führe Schritt 4 für p_i aus. D.h. markiere Zellen, die $B_{r(p_i)}(q)$ schneiden und setze $\alpha'' = r(p_i)$.

(6) return $\alpha = \alpha'' / (1 - \frac{\varepsilon}{2})$

Korrektheit: Es gilt $\alpha'' = r(p_j) \le r$ für geeignetes j.

$$\Rightarrow \alpha \le r / (1 - \tfrac{\varepsilon}{2}) = \frac{r \cdot (1+\varepsilon)}{(1+\varepsilon)(1-\frac{\varepsilon}{2})} = r(1+\varepsilon) \cdot \frac{1}{1 + \varepsilon - \frac{\varepsilon}{2} - \frac{\varepsilon^2}{2}} \le r \cdot (1+\varepsilon), \text{ da } \varepsilon \le 1$$

Weiter gilt $r \le \alpha'' + \sqrt{2}\beta = \alpha \cdot (1 - \frac{\varepsilon}{2}) + \sqrt{2} \frac{\alpha' \varepsilon}{20\sqrt{2}}$

$$= \alpha(1 - \varepsilon/2) + \frac{\alpha'}{10} \cdot \frac{\varepsilon}{2} \le \alpha(1 - \tfrac{\varepsilon}{2}) + r \cdot \tfrac{\varepsilon}{2}$$

$$\Rightarrow r \le \alpha$$

Somit gilt $r \le \alpha \le r(1+\varepsilon)$

Anmerkung: Auch eine Laufzeit von $O(n + K(\frac{1}{\varepsilon^2} + \log n))$ ist möglich, wenn wir in jeder Zelle den Abstand zum nächsten q speichern

Laufzeit: Die Anzahl der zu markierenden Zellen in (4) ist in

$$k \cdot O\left(\left(r/\beta\right)^2\right) = k \cdot O\left(\left(\frac{20\sqrt{2}}{\varepsilon}\right)^2\right) = O\left(\frac{k}{\varepsilon^2}\right)$$

Gleiches gilt falls in Schritt 5 Zellen markiert werden müssen.

Ähnlich wie in A können wir argumentieren, dass maximal ein $p \in \{p_1, \dots, p_{i-1}\}$ als p_i in einer nicht markierten Zelle liegen kann, nämlich das p dessen Zelle den größten Abstand zum nächsten q hat. Wie in (A) gibt eine Rückwärtsanalyse, dass die Laufzeit in $O(n + \frac{k}{\varepsilon^2}) + \sum_{i=2}^{n} O(1 + \frac{k}{\varepsilon^2 i}) = O(n + \frac{k}{\varepsilon^2} \log n)$ ist.