

Finding the Closest Pair of Points:
A Randomized Approach

Thomas Franz

21. Juni 2009

Inhaltsverzeichnis

1	Das Problem	3
2	Entwurf des Algorithmus'	3
2.1	Testen einer vorgegebenen Distanz δ	4
2.2	Eine Datenstruktur zur Verwaltung der Teilquadrate	6
2.3	Der Algorithmus in Pseudocode	7
3	Analyse des Algorithmus'	8
3.1	Vorzeitige Laufzeitanalyse	8
3.2	Beschränkung der erwarteten Anzahl von „Insert“- Operationen	8
3.3	Erwartete Laufzeit des Algorithmus'	9
3.4	Analyse der Wörterbuch-Operationen	9

Bisher wurde für das Problem „Finding the Closest Pair of Points“ ein divide-and-conquer Algorithmus kennengelernt, der das Problem in $O(n \log n)$ Zeit löst.

Im Folgenden soll ein randomisierter Algorithmus vorgestellt werden, welcher mit einer geeigneten Wörterbuch-Datenstruktur das Problem in $O(n)$ erwarteter Zeit, plus $O(n)$ erwarteten Wörterbuchoperationen korrekt löst. Da sich Wörterbücher i.A. sehr effizient mit Hash-Verfahren implementieren lassen, erhalten wir durch die richtige Wahl des Hash-Verfahrens die oben genannte Laufzeit.

1 Das Problem

Gegeben sei eine Menge $P = \{p_1, \dots, p_n\}$ von n Punkten, die allesamt in der euklidischen Ebene liegen, d.h. $p_j = (x_j, y_j) \in \mathbb{R}^2$, und δ bezeichne den Abstand zweier Punkte.

Gesucht ist nun das Punktepaar (p_i, p_j) , welches den kleinsten Abstand bzgl. der euklidischen Metrik hat, also das Paar (p_i, p_j) mit $\delta = \min_{i,j} d(p_i, p_j)$.

Um die Problemstellung zu vereinfachen nimmt man an, dass $p_j \in \{(x_j, y_j) \mid 0 \leq x_j, y_j < 1\}$ für alle $j = 1, \dots, n$. Dies ist keine Einschränkung der Allgemeinheit, denn die Menge P kann in Linearzeit durch Skalierung und Translation in das obige Einheitsquadrat transformiert werden.

2 Entwurf des Algorithmus'

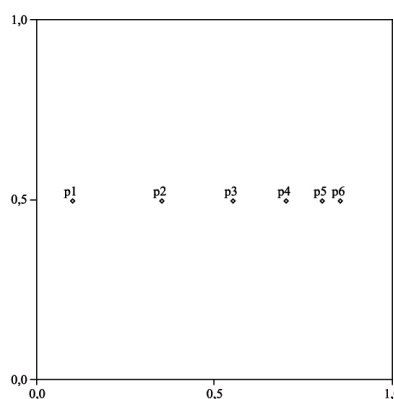
Die Grundidee des Algorithmus' lässt sich wie folgt beschreiben: Die Punkte werden in zufälliger Reihenfolge geordnet und nacheinander, beginnend von vorn, abgearbeitet. Dabei wird die derzeitige, kürzeste Entfernung δ zweier Punkte zwischengespeichert. Erreicht man einen noch nicht betrachteten Punkt p in dieser Folge, wird in seiner „Nachbarschaft“ nach einem schon zuvor besuchten Punkt p' gesucht, der einen geringeren Abstand als δ von p hat. Falls so ein Punkt p' nicht existiert, wird der nächste Punkt in der Folge betrachtet. Findet man jedoch so einen Punkt p' , müssen δ und das aktuelle „Closest Pair“ aktualisiert werden.

Dieses Vorgehen lässt sich wie folgt konkretisieren. Sei p_{i_1}, \dots, p_{i_n} mit $i_j \in \{1, \dots, n\}$ die zufällig geordnete Punktefolge bzw. erhalte durch Umm Nummerierung die Folge p_1, \dots, p_n . Der Algorithmus verläuft in mehreren Phasen, wobei in jeder Phase das „Closest Pair“ nicht geändert wird. In der ersten Phase wird $\delta := d(p_1, p_2)$ gesetzt, also auf den Abstand der ersten beiden Punkte in der Folge. Das Ziel jeder Phase ist es, entweder zu bestätigen, dass δ tatsächlich der Abstand des „Closest Pairs“ ist, oder ein Punktepaar

(p_i, p_j) mit $d(p_i, p_j) < \delta$ zu finden. Während einer Phase werden laufend neue Punkte der Reihe nach hinzugefügt und betrachtet, bis ein Punkt p_i erreicht wird, für den es ein $j < i$ gibt mit $d(p_i, p_j) < \delta$. Dann wird die Phase beendet und für die neue Phase wird $\delta := \min_{j:j < i} d(p_i, p_j)$ gesetzt. Das heißt in der ersten Phase, nachdem $\delta := d(p_1, p_2)$ gesetzt wurde, wird der Punkt p_3 hinzugefügt und dann überprüft, ob $d(p_3, p_1) < \delta$ oder $d(p_3, p_2) < \delta$ gilt. Falls eine der beiden Ungleichungen erfüllt ist, wird die Phase beendet, ansonsten wird p_4 hinzugefügt. Existiert kein derartiger Punkt p_i , so ist das „Closest Pair“ (mit dem Abstand δ) gefunden.

Die Anzahl der Phasen ist offensichtlich abhängig von der zufällig geordneten Punktefolge. Falls p_1, p_2 die am nächsten beieinanderliegenden Punkte sind, genügt eine Phase. Verringert jedoch jeder neu hinzugefügte Punkt die minimale Distanz, so durchläuft der Algorithmus $n - 2$ Phasen (s. Abbildung 1).

Wie wird nun effizient getestet, ob das aktuelle Punktepaar immer noch das „Closest Pair“ ist, nachdem ein weiterer Punkt hinzugefügt wurde, oder, falls nicht, wie wird ein neues Paar mit geringerem Abstand gefunden? Diese Prozedur stellt den Kern des Algorithmus dar.

Abbildung 1: $n - 2$ Phasen

2.1 Testen einer vorgegebenen Distanz δ

Das Einheitsquadrat wird in mehrere kleine Teilquadrate mit Seitenlänge $\frac{\delta}{2}$ zerlegt. Genauer gesagt in N^2 viele Teilquadrate, wobei $N = \lceil \frac{2}{\delta} \rceil$.

Definition 2.1.1. Für $0 \leq s \leq N - 1$ und $0 \leq t \leq N - 1$ definiere S_{st} als

$$S_{st} := \{(x, y) \in \mathbb{R}^2 \mid \frac{s\delta}{2} \leq x < \frac{(s+1)\delta}{2}, \frac{t\delta}{2} \leq y < \frac{(t+1)\delta}{2}\}$$

Durch diese Zerlegung in N^2 Teilquadrate erhält man zwei nützliche Eigenschaften für das Vorhaben:

Lemma 2.1.2. Seien $p, q \in P$. Falls $p \in S_{st}$ und $q \in S_{st} \Rightarrow d(p, q) < \delta$.

Beweis. Seien $p, q \in S_{st}$ mit $p = (p_1, p_2), q = (q_1, q_2)$, dann gilt: $|p_i - q_i| \leq \frac{\delta}{2}$.
 $\Rightarrow d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2} \leq \sqrt{(\frac{\delta}{2})^2 + (\frac{\delta}{2})^2} = \frac{\delta}{\sqrt{2}} < \delta$

□

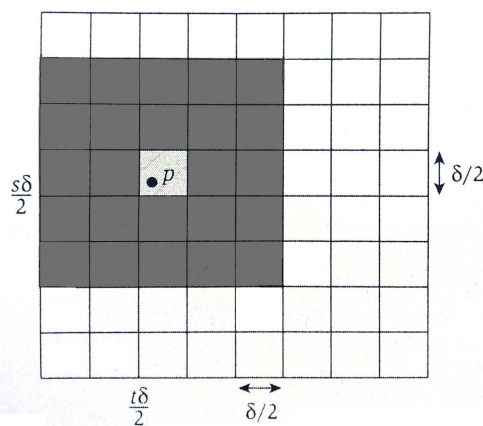


Abbildung 2: Zerlegung des Einheitsquadrates. Der Punkt p liegt im Quadrat S_{st} . Der dunkle Bereich markiert alle Teilquadrate, die *nah* zu S_{st} liegen.

Definition 2.1.3. Zwei Teilquadrate S_{st} und $S_{s't'}$ liegen *nah*, falls $|s - s'| \leq 2$ und $|t - t'| \leq 2$.

Lemma 2.1.4. Sei $p \in S_{st}$ und $q \in S_{s't'}$. Falls $d(p, q) < \delta$, dann liegen S_{st} und $S_{s't'}$ *nah*.

Beweis. Sei $p \in S_{st}$, $q \in S_{s't'}$.

Ann: S_{st} und $S_{s't'}$ nicht *nah*. Somit ist entweder $|s - s'| > 2$ oder $|t - t'| > 2$, d.h. $|p_1 - q_1| \geq \delta$ oder $|p_2 - q_2| \geq \delta$ nach Konstruktion der Zerlegung. Damit folgt (CE $|p_1 - q_1| \geq \delta$):

$$d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2} \geq \sqrt{(\delta)^2 + (p_2 - q_2)^2} \geq \sqrt{(\delta)^2} \geq \delta$$

□

Folgerung: Es genügt die Suche nach einem Punkt p_j , $j \leq i$, in einem 5×5 Umfeld um den derzeitigen Punkt p_i durchzuführen (s. Abbildung 2), denn nur diese Teilquadrate liegen *nah*. Somit müssen maximal 25 Teilquadrate durchsucht werden.

Mithilfe der beiden Lemmata kann der Kern des Algorithmus neu formuliert werden: Sei δ der geringste Abstand der bisher bearbeiteten Punkten p_1, \dots, p_i . Für alle bisherigen Punkte merkt man sich das jeweilige Teilquadrat, in welchem der Punkt liegt. Nun wird p_{i+1} hinzugefügt und geschaut, in welchem Teilquadrat S_{st} der Punkt liegt. Dieses sei $S_{s't'}$. Danach sucht man in einem 5×5 Umfeld von $S_{s't'}$ nach Punkten p_j , $j \leq i$, denn nur diese können den aktuellen Abstand verringern (Lemma 2.1.4). Für jeden gefundenen Punkt p_j , $j \leq i$, wird der Abstand zu p_{i+1} berechnet. Nach Lemma

2.1.2 liegt höchstens ein Punkt in einem Teilquadrat, und somit wird nur eine konstante Anzahl an Distanzberechnungen durchgeführt.

2.2 Eine Datenstruktur zur Verwaltung der Teilquadrate

Sei P' die Menge der bereits besuchten Punkte $p \in P$. Ein weiterer, wichtiger Bestandteil des Algorithmus' ist die Nennung aller Punkte $p \in P$, die in einem bestimmten Quadrat S_{st} liegen.

Zur Implementierung solcher Operationen eignet sich besonders gut ein Wörterbuch. Zu jedem Punkt $p' \in P'$ wird das zugehörige Quadrat S_{st} in das Wörterbuch eingefügt und mit p' indiziert. Jetzt, wenn der nächste Punkt p in der Folge erreicht wird, bestimmt man zuerst das entsprechende Teilquadrat S_{st} und führt dann eine „Lookup“-Operation für alle 25 Teilquadrate, die nah zu S_{st} liegen, aus. Für jeden gefundenen Punkt $p' \in P'$ wird der Abstand $d(p, p')$ berechnet und anschließend mit dem aktuellen Wert δ verglichen. Falls alle berechneten Distanzen größer als δ sind, fügt man S_{st} , indiziert mit p , dem Wörterbuch hinzu.

Wird jedoch ein Punkt $p' \in P'$ gefunden mit $\delta' := d(p, p') < \delta$, so muss das „Closest Pair“ aktualisiert werden. Da sich in diesem Fall der minimale Abstand von δ auf δ' verringert hat, muss die Zerlegung des Einheitsquadrates erneuert werden, damit die oben aufgestellten Lemmata weiterhin ihre Gültigkeit behalten. Dies bedeutet, dass ein neues, leeres Wörterbuch erstellt werden muss, welches Teilquadrate der Seitenlänge $\frac{\delta'}{2}$ verwaltet. Diesen Vorgang nennt man „MakeDictionary“. Für jeden Punkt $p' \in P'$ wird erneut das zugehörige Quadrat S_{st} (der neuen Zerlegung) in das Wörterbuch eingefügt und mit p' indiziert. Nachdem das neue Wörterbuch erstellt wurde, wird der nächste Punkt in der Folge betrachtet.

2.3 Der Algorithmus in Pseudocode

Algorithmus 1 Closest Pairs

```

ordne Punkte in zufälligen Reihenfolge  $p_1, \dots, p_n$ ;
 $\delta := d(p_1, p_2)$ ;
rufe Prozedur Makedictionary auf, um Teilquadrate der Seitenlänge  $\frac{\delta}{2}$  zu
verwalten;
for  $i = 1, 2, \dots, n$  do
  bestimme Teilquadrat  $S_{st}$  mit  $p_i \in S_{st}$ ;
  führe „Lookup“-Operation für alle Teilquadrate, die nah zu  $S_{st}$  liegen,
  aus;
  berechne  $d(p_i, p_j)$  für alle  $p_j$ , die in den nahen Teilquadraten gefunden
  wurden;
  if  $\exists p_j (j < i) : d(p_i, p_j) =: \delta' < \delta$  then
    lösche aktuelles Wörterbuch;
    rufe Prozedur Makedictionary auf, um Teilquadrate der Seitenlänge
     $\frac{\delta'}{2}$  zu verwalten;
    for  $k = 1, 2, \dots, i$  do
      bestimme Teilquadrat  $S_{s't'}$  der Seitenlänge  $\frac{\delta'}{2}$  mit  $p_k \in S_{s't'}$ ;
      füge  $S_{s't'}$ , mit  $p_k$  indiziert, dem neuen Wörterbuch hinzu;
    end for
  else
    füge  $S_{st}$ , mit  $p_i$  indiziert, dem Wörterbuch hinzu;
  end if
end for

```

3 Analyse des Algorithmus'

3.1 Vorzeitige Laufzeitanalyse

Der Algorithmus benötigt eine konstante Anzahl an „Lookup“-Operationen, um einen neuen Punkt p_i zu finden und zudem eine konstante Anzahl an Distanzberechnungen. Weiter wird die „MakeDictionary“-Operation maximal n -mal aufgerufen. Dies trifft zu, wenn sich in jeder Iteration das „Closest Pair“ ändert.

Jetzt fehlt lediglich noch die Analyse der erwarteten Anzahl an „Insert“-Operationen. Aber vorest wird als Zwischenresultat festgehalten:

Der Algorithmus löst das Problem „Finding the Closest Pair“ jeder Zeit korrekt. Dabei werden höchstens $O(n)$ Distanzberechnungen, $O(n)$ „Lookup“-Operationen und $O(n)$ „MakeDictionary“-Operationen ausgeführt.

3.2 Beschränkung der erwarteten Anzahl von „Insert“-Operationen

Sei X eine Zufallsvariable, welche die Anzahl der „Insert“-Operationen bezeichnet. Diese Zufallsvariable ist somit abhängig von der zufällig sortierten Punktefolge.

Die Aufgabe liegt nun darin $E[X]$ zu beschränken. Um dies zu vereinfachen zerlegt man X in n einfachere Zufallsvariablen X_i , welche wie folgt definiert sind:

$$X_i := \begin{cases} 1, & \text{falls der } i\text{-te Punkt in der Folge den min. Abstand ändert} \\ 0, & \text{sonst} \end{cases}$$

Mit diesen Vereinbarungen folgt:

Lemma 3.2.1. *Der Algorithmus führt $X = n + \sum_i iX_i$ „Insert“-Operationen aus.*

Beweis. Es gibt genau n Punkte und jeder wird mindestens einmal in ein Wörterbuch eingefügt $\Rightarrow n$ „Insert“-Operationen.

Weiter werden i Punkte erneut in ein Wörterbuch eingefügt, falls sich der min. Abstand in der i -ten Iteration verringert $\Rightarrow \sum_i iX_i$

$$\Rightarrow X = n + \sum_i iX_i$$

□

Auch die Wahrscheinlichkeit $Pr[X_i = 1]$ kann folgendermaßen beschränkt werden:

Lemma 3.2.2. $Pr[X_i = 1] \leq \frac{2}{i}$

Beweis. Seien p_1, \dots, p_i die ersten i Punkte in der zufälligen Reihenfolge. Nehme an, dass der minimale Abstand unter diesen Punkten durch p und q erlangt wird. Dann kann p_i den min. Abstand nur verringern, falls $p_i = p$ oder $p_i = q$. Da aber die ersten i Punkte zufällig geordnet wurden, könnte jeder von ihnen mit gleicher Wahrscheinlichkeit der Letzte sein, somit ist die Wahrscheinlichkeit, dass p oder q der i -te Punkt ist, $\frac{2}{i}$. Unter den ersten i Punkten kann es jedoch mehrere Paare geben, welche den selben geringsten Abstand haben. Somit gilt also: $Pr[X_i = 1] \leq \frac{2}{i}$. □

Zusammen ergibt sich eine obere Schranke für die erwartete Anzahl von „Insert“-Operationen:

$$\begin{aligned} E[X] &= E\left[n + \sum_i iX_i\right] \\ &= n + \sum_i iE[X_i] \\ &\leq n + \sum_i i\frac{2}{i} \\ &= n + 2n \\ &= 3n \end{aligned}$$

3.3 Erwartete Laufzeit des Algorithmus'

Kapitel 3.1 und Kapitel 3.2 ergeben:

Der randomisierte „Closest-Pair“ Algorithmus benötigt eine Laufzeit von $O(n)$, sowie $O(n)$ Wörterbuch-Operationen im Erwartungswert.

3.4 Analyse der Wörterbuch-Operationen

Bisher wurde die Wörterbuch-Datenstruktur als eine Art „Black Box“ betrachtet, und die Laufzeit wurde im Erwartungswert durch $O(n)$ plus zusätzliche $O(n)$ Wörterbuch-Operationen beschränkt.

Diese Wörterbuch-Operationen sollen nun präzisiert werden, damit man die tatsächliche, erwartete Laufzeit beschränken kann.

Zur Implementierung des Wörterbuchs wird ein universelles Hash-Verfahren verwendet. Sobald nun ein neuer Punkt p_i dem Wörterbuch hinzugefügt wird, nutzt der Algorithmus die Hash-Tabellen „Lookup“-Operation, um alle Punkte in den 25 Teilquadraten, die *nahe* zu p_i liegen, zu finden. Hat die Hash-Tabelle Kollisionen, dann werden die 25 „Lookup“-Operationen mehr als 25 Punkte betrachten. Jedoch gilt, dass jede „Lookup“-Operation im Erwartungswert nur $O(1)$ zuvor eingefügte Punkte betrachtet.

Intuitiv liegt eine erwartete Laufzeit von $O(n)$ nahe. Da sich aber die zwei unterschiedlichen Zufallskomponenten (zufällige Ordnung der Punkte und erstellen einer Hash-Tabelle mithilfe einer universellen Hash-Funktion) einander beeinflussen, wird diese Intuition durch ein Lemma festgehalten:

Lemma 3.4.1. *Wird der „Closest Pair“ Algorithmus mit einem universellen Hash-Verfahren implementiert, so ist die Anzahl der betrachteten Punkte durch die „Lookup“-Operationen im Erwartungswert durch $O(n)$ beschränkt.*

Beweis. Sei X eine Zufallsvariable, welche die Anzahl der durch den Algorithmus aufgerufenen „Lookup“-Operationen bezeichnet, und sei σ die durch den Algorithmus zufällig geordnete Punktefolge. Die Ordnung σ bestimmt daher die Abfolge der von dem Algorithmus betrachteten, minimalen Abstandswerten und die Abfolge der durchzuführenden Wörterbuchoperationen. Somit bestimmt die Wahl von σ den Wert von X . Nenne diesen Wert $X(\sigma)$ und bezeichne mit ε_σ das Ereignis, das der Algorithmus σ als zufällige Ordnung wählt. Daher gilt $E[X|\varepsilon_\sigma] = X(\sigma)$.

Nach Kapitel 3.3 gilt: $E[X] \leq c_0 n$ mit $c_0 \in \mathbb{R}$ konstant. Nun betrachte man die Folge der „Lookup“-Operationen für eine gegebene Ordnung σ . Für $i = 1, \dots, X(\sigma)$, sei Y_i die Anzahl der Punkte, welche während der i -ten „Lookup“-Operation betrachtet werden. Also die Anzahl der zuvor eingefügten Punkte, welche mit dem Wörterbucheintrag dieser „Lookup“-Operation kollidieren. Es gilt: $E[Y_i|\varepsilon_\sigma] = O(1)$, bzw. $E[Y_i|\varepsilon_\sigma] \leq c_1$ mit $c_1 \in \mathbb{R}$ konstant, für alle beliebigen Ordnungen σ und für alle i .

$$\Rightarrow E \left[\sum_i Y_i | \varepsilon_\sigma \right] \leq c_1 X(\sigma)$$

Damit kann nun $\sum_{i=1}^{X(\sigma)} Y_i$ beschränkt werden:

$$\begin{aligned}
 E \left[\sum_{i=1}^{X(\sigma)} Y_i \right] &= \sum_{\sigma} Pr[\varepsilon_{\sigma}] \cdot E \left[\sum_i Y_i | \varepsilon_{\sigma} \right] \\
 &\leq \sum_{\sigma} Pr[\varepsilon_{\sigma}] \cdot c_1 X(\sigma) \\
 &= c_1 \sum_{\sigma} Pr[\varepsilon_{\sigma}] \cdot E[X | \varepsilon_{\sigma}] \\
 &= c_1 E[X] \\
 &\leq c_0 c_1 n \\
 &= O(n)
 \end{aligned}$$

□

Verwendet man zur Implementierung eine Familie von universellen Hash-Funktionen, so werden im Erwartungswert also höchstens $O(n)$ Hash-Funktionswerte berechnet. Wird zudem immer dieselbe Hashtabelle der Größe $p \geq n$ (p prim und fest) benutzt, so kann man für die endgültige Laufzeit festhalten:

Der randomisierte „Closest-Pair“ Algorithmus berechnet im Erwartungswert $O(n)$ Hash-Funktionswerte und benötigt zusätzlich $O(n)$ Zeit, um das „Closest-Pair“ zu bestimmen.

Eine Frage steht noch offen: Welche Zeit wird für die Auswertung einer Hash-Funktion benötigt? Unter der Verwendung eines komplexen Hash-Verfahrens ist es möglich, die arithmetischen Operationen in $O(1)$ durchzuführen. Jedoch benötigt so ein Verfahren eine Primzahl p , die Größer ist als N^2 , also größer als die Anzahl der Teilquadrate. Da aber die Anzahl der Teilquadrate inverse zu dem minimalen Abstand wächst, benötigt man jedes mal, sobald sich der minimale Abstand verringert, eine neue Primzahl p und somit auch eine neue Hashtabelle. Es ist jedoch möglich, diese Problem zu bewältigen, und schließlich erreicht man eine Gesamtlaufzeit von $O(n)$ im Erwartungswert.