Distanzerhaltende Approximation von Kantenzügen

Nikolas Klug

Sommersemester 2018
Universität Augsburg
Seminar Algorithmen und Datenstrukturen

Zusammenfassung

Diese Seminararbeit basiert auf „Distance-preserving approximations of polygonal paths“ von J. Gudmundsson et al. (*Computational Geometry 36*, S. 183-196, 2007). Sei $P = (p_1, p_2, \dots, p_n)$ ein Kantenzug und $t \geq 1$. Ein Kantenzug Q approximiert P, falls er nur aus Punkten von P besteht und die Längen seiner Kanten höchstens um einen Faktor t von P abweichen. Diese Arbeit stellt exakte und approximative Algorithmen vor, um Q so zu berechnen, dass Q für festes t die minimale Zahl von Knoten besitzt oder für eine feste Knotenzahl minimales t hat.

1 Einführung und Definitionen

Ein *(polygonaler) Kantenzug* $P = (p_1, p_2, \dots, p_n)$ ist eine Aneinanderreihung von Geradensegmenten, die für $1 \leq i < n$ jeweils die Punkte p_i und p_{i+1} verbinden. Die Punkte p_i stammen dabei aus $\mathbb{R}^d$ (für $d \in \mathbb{N}$). Aufgrund der starken Ähnlichkeit zu Pfaden in (gerichteten) Graphen werden wir für Kantenzüge auch häufig Begriffe aus der Graphentheorie verwenden. Beispielsweise heißen die Punkte des Kantenzuges auch Knoten, die verbindenden Geraden auch (gerichtete) Kanten etc. In manchen Vorlesungen und Fachbüchern wurden und werden die Begriffe Kantenzug und Pfad teilweise synonym verwendet. Hier wollen wir sie aber – der besseren Verständlichkeit wegen – klar unterscheiden.

In der Realität kann ein Kantenzug mitunter eine hohe Zahl von Knoten besitzen, die für die Anwendung in einer solchen Menge nicht benötigt werden. Hier ist es vorteilhaft, den Kantenzug durch einen ähnlichen Kantenzug mit deutlich weniger Kanten (und somit auch Knoten) zu approximieren, wobei sich wichtige Parameter allerdings nicht stark ändern sollten. In der Fachliteratur werden dabei zahlreiche verschiedene Parameter aufgeführt, die bei der Approximation erhalten werden sollen, einige davon sind die Fläche [1], die der Pfad einschließt und die Distanz bzw. die Länge des Pfades [4].

Ein Anwendungsszenario der Kantenzugapproximation liegt zum Beispiel in der Erstellung und Vereinfachung von Karten. Dort trifft man ständig auf polygonale Kantenzüge im zwei-, selten auch dreidimensionalen Raum: Straßen, Küstenlinien, Höhenlinien, Stadtumrisse und Landesgrenzen sind nur einige Beispiele. Leider sind in den meisten Fällen nicht alle dieser Kantenzüge so einfach darzustellen wie beispielsweise die Grenze zwischen Libyen und Ägypten. Betrachtet man die Weltkarte, fällt auf, dass die meisten Grenzen sogar sehr komplizierte Formen annehmen, die aus tausenden, wenn nicht zehntausenden einzelnen Punkten und Kanten bestehen. Möchte man nun eine Karte erstellen, ist klar, dass es im Allgemeinen nicht möglich und auch nicht notwendig ist, alle dieser Punkte und Kanten darzustellen. Genau hier liegt eine Anwendung der Kantenzugapproximation: Die Landesgrenze bzw. die anderen oben aufgeführten Beispiele werden durch einen

Kantenzug approximiert, der aus deutlich weniger Punkten besteht, ohne dass sich ein wichtiger Parameter ändert, wie zum Beispiel die Länge.

In dieser Arbeit soll es um Algorithmen gehen, die einen polygonalen Kantenzug unter ungefährer Einhaltung der Länge approximieren. Dabei betrachten wir zunächst zwei exakte Algorithmen und später noch zwei approximative, die eine deutliche bessere Laufzeit aufweisen als die exakten.

Bevor wir uns den Algorithmen zuwenden, müssen wir jedoch einige Grundbegriffe definieren.

1.1 Definitionen

Sei $P = (p_1, p_2, \ldots, p_n)$ ein Kantenzug und seien $u, v \in \mathbb{R}^d$. Wir definieren $|uv|$ als den euklidischen Abstand von u und v. Sind p_i, p_j Knoten von P (im Folgenden kurz $p_i, p_j \in P$), dann ist $\delta(p_i, p_j) := \sum_{k=i}^{j-1} |p_k p_{k+1}|$, also der euklidische Abstand dieser beiden Punkte entlang des Pfades P.

Lemma 1.1. *Für alle Knoten $p_i, p_j \in P$ gilt $|p_i p_j| \leq \delta(p_i, p_j)$*

Beweis. Die Strecke zwischen zwei Punkten in $\mathbb{R}^d$ ist immer die kürzeste Verbindung dieser beiden Punkte. □

Bis jetzt haben wir nur über distanzerhaltende Approximationen eines Kantenzugs geredet, ohne festzulegen, was distanzerhaltend eigentlich bedeutet. Wir nennen eine Kante distanzerhaltend, wenn ihre Länge nur um einen bestimmten Faktor von der Länge des Kantenzugs abweicht. Genauer:

Definition 1.2 (t-distanzerhaltend)**.** *Seien $t \in \mathbb{R}$, $t \geq 1$ und $p_i, p_j \in P$. Dann ist die Kante (p_i, p_j) genau dann* t-distanzerhaltend*, wenn $\delta(p_i, p_j) \leq t \cdot |p_i p_j|$.*

Damit können wir jetzt definieren, was eine distanzerhaltende Approximation ist.

Definition 1.3 (t-distanzerhaltende Approximation)**.** *Ein Kantenzug $Q = (p_{i_1}, p_{i_2}, \ldots, p_{i_k})$ ist genau dann eine* t-distanzerhaltende Approximation von $P = (p_1, p_2, \ldots, p_n)$*, wenn beide der folgenden Bedingungen gelten.*

(1) $1 = i_1 < i_2 < \ldots < i_k = n$.

(2) Für alle $1 \leq l < k$ ist die Kante $(p_{i_l}, p_{i_{l+1}})$ des Kantenzugs t-distanzerhaltend.

Wir nennen eine t-distanzerhaltende Approximation eines Pfades *minimal*, falls sie die geringst mögliche Zahl an Knoten besitzt. Der Quotient $\frac{\delta(p_i, p_j)}{|p_i p_j|}$ heißt *Abweichung* der Kante (p_i, p_j) vom Kantenzug.

Korollar 1.4. *Sei $1 \leq t < t'$. Dann ist jede t-distanzerhaltende Approximation eines Pfades P auch eine t'-distanzerhaltende Approximation von P.*

In Zusammenhang mit der distanzerhaltenden Kantenzugapproximation stellen sich im Wesentlichen die folgenden zwei Probleme:
Das **Minimum-Vertex-Path-Simplification Problem (MVPS)**: Liegt ein polygonaler Kantenzug P und eine reelle Zahl $t \geq 1$ vor, soll eine kürzeste t-distanzerhaltende Approximation von P berechnet werden.
Das **Minimum-Dilation-Path-Simplification Problem (MDPS)**: Liegt ein polygonaler Kantenzug P und eine natürliche Zahl k vor, soll der kleinste Wert t bestimmt werden, für den eine t-distanzerhaltende Approximation von P mit maximal k Knoten existiert.

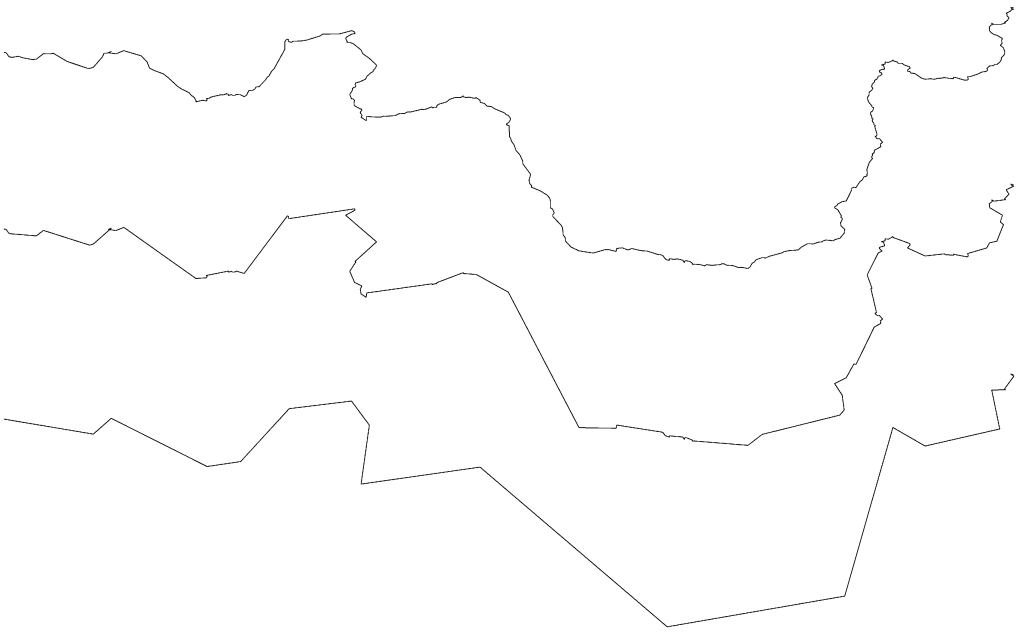

Abb. 1: Kantenzug mit 430 Punkten (oben) und zwei Approximationen mit 126 bzw. 22 Punkten (Mitte und unten), die aus dem approximativen Algorithmus für das MVPS-Problem mit jeweils $\epsilon = 0.05$ und $t = 1.05$ bzw. $t = 1.2$ berechnet wurden (Quelle: [4])

Es mag verwunderlich sein, dass beim MDPS-Problem nur nach dem kleinsten t-Wert, nicht aber nach einer dazugehörigen t-distanzerhaltenden Approximation gefragt ist. Eine solche können wir aber erhalten, indem wir für vorliegendes t das MVPS-Problem lösen – was allerdings, wie wir sehen werden, nichts an der asymptotischen Laufzeit ändert.

2 Exakte Algorithmen für MVPS und MDPS

Zu Beginn betrachten wir zwei einfache exakte Algorithmen. Seien wieder $d \in \mathbb{N}$, $t \geq 1$ und $P = (p_1, p_2, \ldots, p_n)$ ein Kantenzug in $\mathbb{R}^d$. Sei weiter P^* die Menge aller minimalen t-distanzerhaltenden Approximationen von P.

Wir konstruieren jetzt den gerichteten Graphen $G_t = (V, E_t)$, wobei V genau aus den Knoten des Pfades P besteht und $E_t = \{(p_i, p_j) \in V \times V |\ i < j$ und (p_i, p_j) ist t-distanzerhaltend$\}$. E_t ist also gerade die Menge aller t-distanzerhaltenden Kanten zwischen Knoten aus V. Zunächst beobachten wir, dass jede t-distanzerhaltende Approximation von P einem Pfad in G_t entspricht, da G_t alle t-distanzerhaltenden Kanten zwischen Knoten von P enthält. Andererseits ist auch jeder Pfad $Q = (p_{i_1}, p_{i_2}, \ldots, p_{i_k})$ mit $1 = i_1 < i_2 < \ldots < i_k = n$ in G_t eine t-distanzerhaltende Approximation von P, da nur t-distanzerhaltende Kanten verwendet werden. Daraus folgt, dass auch jeder Kantenzug aus P^* in G_t liegt. Jetzt müssen wir also nur noch ein Element von P^* ermitteln. Das ist leicht: Wir führen eine Breitensuche in G_t mit Startknoten p_1 durch, bei der wir jeden Knoten mit der Nummer des Knotens beschriften, von dem aus er zum ersten Mal entdeckt wurde (also mit der Nummer seines Vaters im BFS-Baum). Am Ende lesen wir diese Beschriftung bei p_n beginnend solange aus, bis wir p_1 erreichen. Der dadurch entstandene Pfad entspricht dann aufgrund der Eigenschaften der Breitensuche einem Kantenzug aus P^*. Nun betrachten wir noch die Laufzeit: Die Konstruktion von G_t gelingt uns in $O(n^2)$ Zeit, da wir für maximal $\binom{n}{2} = O(n^2)$ Kanten überprüfen müssen, ob diese t-distanzerhaltend sind. Sei m die Zahl der Kanten in G_t. Dann wissen wir aus dem Informatik III Skript [5], dass die Breitensuche $O(n + m)$ Zeit benötigt.

In unserem Fall (und auch sonst immer) ist $m = O(n^2)$, und somit kostet die Breitensuche $O(n^2)$ Zeit. Wir können festhalten:

Satz 2.1. *Das Minimum-Vertex-Path-Simplification Problem kann für Kantenzüge mit n Knoten in $O(n^2)$ Zeit gelöst werden.*

Als Nächstes wollen wir uns überlegen, wie man das MDPS-Problem für eine feste Anzahl von Knoten k lösen kann. Sei im Folgenden κ_t die geringst mögliche Zahl von Knoten für eine t-distanzerhaltende Approximation von P.

Lemma 2.2. *Sind $t, t' \in \mathbb{R}$ und $1 \leq t < t'$, dann ist $\kappa_t \geq \kappa_{t'}$.*

Beweis. Wäre $\kappa_t < \kappa_{t'}$, hätte eine minimale t-distanzerhaltende Approximation von P echt weniger Knoten als eine minimale t'-distanzerhaltende. Aber jede t-distanzerhaltende Approximation von P ist nach Korollar 1.4 auch eine t'-distanzerhaltende Approximation von P. Das ist ein Widerspruch. □

Da G_t maximal $O(n^2)$ Kanten enthält, gibt es eine endliche Zahl von t-Werten, die eine Approximation von P exakt annehmen kann (das sind nämlich genau die Abweichungen der Kanten aus G_t von P). Wir müssen also nur noch aus diesen Werten den geringsten Wert t^* ermitteln, für den eine t^*-distanzerhaltende Approximation existiert, die gerade noch k Knoten oder weniger hat. Dazu definieren wir zunächst $t^*_{ij} \coloneqq \frac{\delta(p_i,p_j)}{|p_i p_j|}$ für $1 \leq i < j \leq n$ als die Abweichung der Kante (p_i, p_j) vom Pfad. Sei nun $M \coloneqq \{t^*_{ij} \mid 1 \leq i < j \leq n\}$. Wir wissen, dass $t^* \in M$, da die gesuchte Approximation eine Kante mit maximalem t-Wert hat, und in M gerade alle für diese Kante möglichen Werte enthalten sind. Aus Lemma 2.2 wissen wir außerdem, dass die κ_t monoton fallen, wenn die t-Werte monoton wachsen. Sortieren wir jetzt M zu M', können wir in M' nach t^* suchen. Da M $O(n^2)$ Elemente enthält, können wir M bekanntermaßen (siehe [5]) in $O(n^2 \log n^2) = O(n^2 \log n)$ sortieren. Für die Suche verwenden wir eine Binärsuche, bei der wir jeweils für den aktuell betrachteten t-Wert das MVPS-Problem lösen und dann abhängig vom Ergebnis entweder im rechten oder linken Teilbereich weitersuchen. Eine gewöhnliche Binärsuche kostet $O(\log n)$ Zeit und mit Satz 2.1 ergibt sich auch hierfür eine Laufzeit von $O(n^2 \log n)$. Wir halten fest:

Satz 2.3. *Das Minimum-Dilation-Path-Simplification Problem kann für Kantenzüge mit n Knoten in $O(n^2 \log n)$ Zeit gelöst werden.*

Unter Verwendung des im Beweis von Satz 2.3 angegebenen Algorithmus könnten wir jetzt zusätzlich für gegebenes t^* eine t^*-distanzerhaltende Approximation von P bestimmen. Das würde uns sogar nur $O(n^2)$ Zeit kosten und wir hätten insgesamt in $O(n^2 \log n)$ Zeit das MDPS-Problem gelöst und eine passende t^*-distanzerhaltende Approximation berechnet.

Mit dieser Erkenntnis beschließen wir das Kapitel über exakte Algorithmen für das MVPS- und das MDPS-Problem und wenden uns approximativen Lösungen zu.

3 Approximative Algorithmen

Die beiden approximativen Algorithmen, die wir betrachten werden, basieren auf der sogenannten Zerlegung in wohl-separierte Paare. Dabei werden im weitesten Sinne die Punkte des Kantenzugs in verschiedenen Mengen zusammengefasst, die bestimmte Eigenschaften haben. Da diese Zerlegung einen wesentlichen Teil beider Algorithmen ausmacht, werden wir sie zunächst genauer definieren und einen Algorithmus zu ihrer Berechnung angeben.

3.1 Well-separated Pair Decomposition

Definition 3.1 (wohl-separiert)**.** *Seien $s > 0$ und A und B zwei endliche Mengen von Punkten in $\mathbb{R}^d$. A und B heißen* wohl-separiert bezüglich s *(engl. well-separated), falls es zwei disjunkte Bälle C_A und C_B gibt, die denselben Radius R haben, sodass $A \subseteq C_A$ und $B \subseteq C_B$ und die euklidische Distanz zwischen den Rändern von C_A und C_B mindestens $s \cdot R$ beträgt.*

Eine solche Zahl s nennen wir die Trennungsrate *der Mengen A und B.*

Das folgende Lemma hält zwei wichtige Eigenschaften von zwei bezüglich s wohl-separierten Mengen A und B fest.

Lemma 3.2. *Seien $a, a' \in A$ und $b, b' \in B$. Dann gilt:*

(1) $|aa'| \leq \frac{2}{s} \cdot |a'b'|$.

(2) $|a'b'| \leq (1 + \frac{4}{s}) \cdot |ab|$.

Beweis. Zu 1. Ist R der Radius von C_A und C_B, so gilt $|aa'| \leq 2 \cdot R$. Da A und B wohl-separiert sind, gilt zudem $|a'b'| \geq s \cdot R$, was äquivalent ist zu $R \leq \frac{|a'b'|}{s}$. Durch Einsetzen folgt dann die Behauptung.

Zu 2. Da A und B bezüglich s wohl-separiert sind und C_A und C_B beide denselben Radius R haben, gilt $|a'b'| \leq s \cdot R + 4 \cdot R$. Ausklammern rechts ergibt $(1 + \frac{4}{s}) \cdot s \cdot R$. Da ja auch $s \cdot R \leq |ab|$, folgt durch Einsetzen die Behauptung. □

Definition 3.3 (WSPD)**.** *Sei $S \subseteq \mathbb{R}^d$ und $s > 0$. Eine Menge $\{(A_1, B_1), (A_2, B_2), \ldots, (A_m, B_m)\}$ (im Folgenden auch $\{(A_i, B_i)\}_{1 \leq i \leq m}$) von Paaren von nicht-leeren Teilmengen von S ist genau dann eine* Zerlegung in wohl-separierte Paare (engl. well-separated pair decomposition; WSPD)*, wenn für alle $1 \leq i \leq m$ gilt:*

(1) $A_i \cap B_i = \emptyset$.

(2) Für alle $p, q \in S$ gibt es genau einen Index $1 \leq j \leq m$, sodass entweder $p \in A_j$ und $q \in B_j$ oder $q \in A_j$ und $p \in B_j$.

(3) A_i und B_i sind bezüglich s wohl-separiert.

m nennen wir dabei die *Größe* der WSPD.

Die WSPD bildet eine wichtige Grundlage für die beiden Algorithmen, die wir im Folgenden betrachten werden. Callahan und Kosaraju haben in [2] gezeigt, dass man zu einer gegebenen Menge $L \subset \mathbb{R}^d$ der Größe n eine WSPD der Größe $m = O(s^d n)$ in $O(n \log n + s^d n)$ Zeit berechnen kann. Dabei wird zunächst in $O(n \log n)$ ein sogenannter *(fairer) Split-Tree* berechnet. Bei diesem handelt es sich um einen binären Baum, in dessen Blättern die Werte der Grundmenge S in von links nach rechts aufsteigend sortierter Reihenfolge gespeichert sind (Abb. 3). Aus dem Split-Tree kann dann in $O(s^d n)$ Zeit eine WSPD erstellt werden. Wir werden sehen, dass es für unsere Anwendung genügt, eine WSPD für Mengen von Punkten aus $\mathbb{R}$ zu erstellen. Für diesen eindimensionalen Fall kann ein fairer Split-Tree mit Hilfe eines einfachen Algorithmus berechnet werden (Abb. 2).

Sei S nun eine endliche Teilmenge von $\mathbb{R}$ und $|S| = n$. Wir können davon ausgehen, dass uns diese Menge sortiert in einem Array $S[1..n]$ vorliegt und werden später sehen, dass das bei unserem Algorithmus auch tatsächlich der Fall ist. Sei T der Split-Tree, der durch das Ausführen von `compute_split_tree(1, n)` entstanden ist. Für jeden inneren Knoten k, also für jeden Knoten, der kein Blatt ist, wird in `compute_split_tree()` zusätzlich das kleinste Intervall der Indizes der

```
compute_split_tree(i, j) {
  if i = j then
    erstelle neuen Knoten u;
    speichere das Intervall [i, i] zu u;
    return u
  else
    z := (S[i] + S[j])/2;
    k := Index eines Elementes von S, sodass S[k] ≤ z < S[k + 1];
    v := compute_split_tree(i, k);
    w := compute_split_tree(k+1, j);
    erstelle neuen Knoten u;
    speichere das Intervall [i, j] zu u;
    mache v zum linken Kind von u;
    mache w zum rechten Kind von u;
    return u
  end if
}
```

Abb. 2: Algorithmus zum Erstellen eines fairen Split-Trees zu einer gegebenen Menge S (nach [4]).

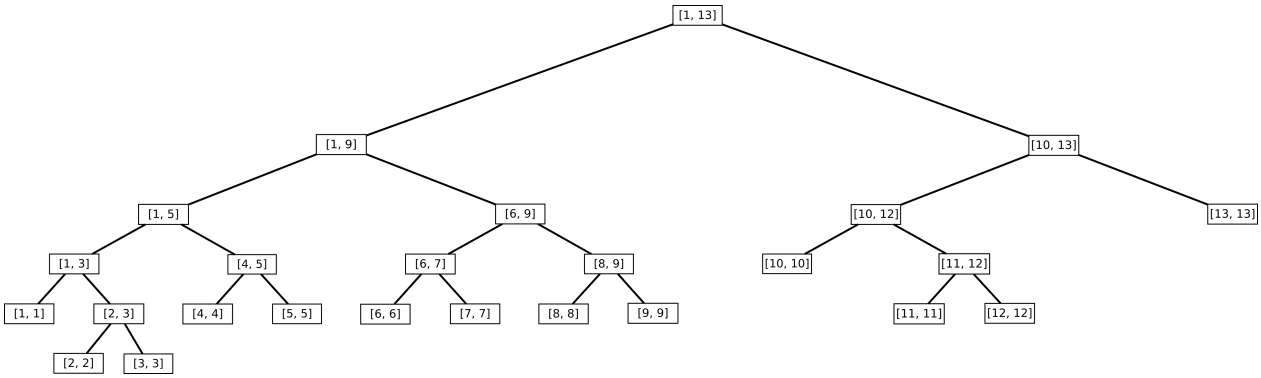

Abb. 3: Fairer Split-Tree für die Menge $S = [0.0, 5.0, 9.1, 17.2, 32.2, 37.6, 44.3, 54.3, 67.9, 81.0, 95.4, 96.4, 141.5]$. Die Knoten des Baumes sind mit dem Intervall beschriftet, das mit ihnen gespeichert ist.

Elemente gespeichert, die die Blätter des von k induzierten Teilbaumes bilden. Die Wurzel speichert also zum Beispiel das Intervall $[1, n]$ und das i-te Blatt das Intervall $[i, i]$. Da T n Blätter hat, erstellen wir insgesamt $O(n)$ Knoten. Dabei müssen wir aber für die Bestimmung von k jedes Mal eine Binärsuche durchführen, die $O(\log n)$ Zeit kostet.

Somit ergibt sich für das Erstellen von T eine Gesamtlaufzeit von $O(n \log n)$. Betrachten wir jetzt zwei innere Knoten p und q von T. Seien $[i, j]$ und $[k, l]$ die Intervalle, die wir mit p und q gespeichert haben und

$$R := \max(S[j] - S[i], S[l] - S[k]).$$

Nach Definition 3.1 sind die beiden Intervalle genau dann wohl-separiert, wenn

$$S[k] - S[j] \geq R \cdot s \text{ oder } S[i] - S[l] \geq R \cdot s.$$

Der in Abbildung 4 dargestellte Algorithmus berechnet dann aus dem eben erstellten fairen Split-Tree T eine WSPD. Betrachten wir diesen nun etwas genauer.

Beim Aufruf von `compute_wspd(T, s)` werden für jeden Knoten k dessen Kindknoten v und w betrachtet, und dann `find_pairs(v, w)` aufgerufen. Da die Elemente von S in den Blättern von T gespeichert sind und jedes Element genau durch ein Blatt dargestellt wird, ist klar, dass der linke Kindknoten v und der rechte w disjunkte Teilmengen von S repräsentieren. Somit ist Forderung (1) der WSPD erfüllt. `find_pairs(v, w)` überprüft, ob die mit v und w gespeicherten Intervalle S_v und S_w wohl-separiert sind; ist dies der Fall, speichern wir mit v, dass seine Blätter das Element

```
compute_wspd(T, s) {
   for each innerer Knoten u in T do
      v := linkes Kind von u;
      w := rechtes Kind von u;
      find_pairs(v, w);
}

find_pairs(v, w) {
   if S_v und S_w sind bezüglich s nicht wohl-separiert then
      Seien [i, j] und [k, l] die Intervalle die mit v bzw. w gespeichert sind;
      if S[j] − S[i] ≤ S[l] − S[k] then
         w_1 := linkes Kind von w;
         w_2 := rechtes Kind von w;
         find_pairs(v, w_1);
         find_pairs(v, w_2);
      else
         v_1 := linkes Kind von v;
         v_2 := rechtes Kind von v;
         find_pairs(v_1, w);
         find_pairs(v_2, w);
      end if
   else
      Speichere in v und w, dass deren Blätter die Teilmengen A und B einer WSPD bilden;
   end if
}
```

Abb. 4: Algorithmus zum Erstellen einer WSPD aus einem gegebenen Split-Tree T und einer Trennungsrate s (nach [4])

A_i einer WSPD bilden, und mit w, dass seine Kinder das Element B_i darstellen. Sind die Intervalle nicht wohl-separiert, steigen wir solange in Richtung des größeren Intervalls im Baum herab, bis wir auf zwei wohl-separierte Intervalle treffen. Das ist spätestens dann der Fall, wenn wir zwei Blätter betrachten, denn dann ist $R = 0$. Wir sehen also, dass die erste und die dritte Forderung der Definition der WSPD durch den Algorithmus erfüllt werden. Man kann auch zeigen, dass er die zweite erfüllt, was wir an dieser Stelle allerdings überspringen werden. Interessierte können einen vollständigen Beweis auf Seite 72ff in [2] nachlesen. Callahan und Kosaraju, die Autoren dieses Artikels, haben auch bewiesen, dass `compute_wspd(T, s)` eine zu T gehörende WSPD der Größe $O(sn)$ in $O(sn)$ Zeit berechnet. Halten wir also fest:

Satz 3.4. *Sei $S \subset \mathbb{R}$ endlich und $n = |S|$. Dann kann in $O(n \log n + sn)$ Zeit ein Split-Tree T und eine dazugehörige WSPD $\{(A_i, B_i)\}_{1 \le i \le m}$ der Größe $m = O(sn)$ berechnet werden.*

3.2 Algorithmus für das MVPS-Problem

Nachdem wir jetzt einige Vorarbeit geleistet haben, werden wir in diesem Kapitel sehen, wie man das MVPS-Problem mit Hilfe einer WSPD bis auf ein ϵ genau approximieren kann – und das (für festes t und ϵ) in $O(n \log n)$ Zeit.

Die Theorie

Sei $P = (p_1, p_2, \ldots, p_n)$ ein Kantenzug in $\mathbb{R}^d$. Für unseren Anwendungsfall genügt es, eine eindimensionale Version $S = (x_1, x_2, \ldots, x_n)$ dieses Kantenzugs zu betrachten. Diese erhalten wir, indem wir $x_i = \delta(p_1, p_i)$ setzen (für alle $1 \le i \le n$). Als Nächstes berechnen wir für ein festes $s > 0$ zunächst den Split-Tree T und danach eine zugehörige WSPD $\{(A_i, B_i)\}_{1 \le i \le m}$ von S. Wegen Eigenschaft (3) der WSPD sind für alle $1 \le i \le m$ entweder alle Elemente in A_i kleiner als die in B_i oder umgekehrt.

Wir werden o.B.d.A annehmen, dass alle Elemente, die in A_i enthalten sind, kleiner sind als alle Elemente in B_i, da wir einfach bei der Erstellung der WSPD die beiden Mengen passend benennen.

Lemma 3.5. *Seien $p, p', q, q' \in P$ und sei i ein solcher Index, dass für $x = \delta(p_1, p)$, $x' = \delta(p_1, p')$, $y = \delta(p_1, q)$ und $y' = \delta(p_1, q')$ $x, x' \in A_i$ und $y, y' \in B_i$ sind. Ist weiter $1 \leq t < \frac{s^2}{4s+16}$ und ist die Kante (p, q) t-distanzerhaltend, dann ist (p', q') t'-distanzerhaltend, wobei*

$$t' = \frac{(1 + \frac{4}{s}) \cdot t}{1 - 4(1 + \frac{4}{s}) \cdot \frac{t}{s}}.$$

Beweis. Wegen unserer speziellen Wahl von t ist der Nenner von t' immer positiv, und genauso der Zähler. Dann können wir rechnen:

$$\begin{aligned}
\delta(p', q') &= |x'y'| \\
&\leq (1 + \frac{4}{s}) \cdot |xy| && \text{(Lemma 3.2 (2))} \\
&= (1 + \frac{4}{s}) \cdot \delta(p, q) \\
&\leq (1 + \frac{4}{s})t \cdot |pq| && ((p, q) \text{ ist } t\text{-distanzerhaltend}) \\
&\leq (1 + \frac{4}{s})t \cdot (|pp'| + |p'q'| + |q'q|) && \text{(Dreiecksungleichung in } \mathbb{R}) \\
&\leq (1 + \frac{4}{s})t \cdot (\delta(p, p') + |p'q'| + \delta(q', q)) && \text{(Lemma 1.1)} \\
&= (1 + \frac{4}{s})t \cdot (|xx'| + |p'q'| + |yy'|) \\
&\leq (1 + \frac{4}{s})t \cdot (\frac{2}{s} \cdot |x'y'| + |p'q'| + \frac{2}{s} \cdot |x'y'|) && \text{(Lemma 3.2 (1))} \\
&= (1 + \frac{4}{s})t \cdot (\frac{4}{s} \cdot \delta(p', q') + |p'q'|) \\
&= 4(1 + \frac{4}{s})\frac{t}{s} \cdot \delta(p', q') + (1 + \frac{4}{s})t \cdot |p'q'|
\end{aligned}$$

Damit ergibt sich die folgende Äquivalenzkette:

$$\begin{aligned}
& \delta(p', q') \leq 4(1 + \frac{4}{s})\frac{t}{s} \cdot \delta(p', q') + (1 + \frac{4}{s})t \cdot |p'q'| \\
\Leftrightarrow \quad & \delta(p', q') \cdot (1 - 4(1 + \frac{4}{s})\frac{t}{s}) \leq (1 + \frac{4}{s})t \cdot |p'q'| \\
\Leftrightarrow \quad & \delta(p', q') \leq t' \cdot |p'q'|
\end{aligned}$$

□

Sei jetzt $0 < \epsilon < \frac{1}{3}$ und $1 \leq t$. Sei

$$s = \frac{12 + 24(1 + \frac{\epsilon}{3}) \cdot t}{\epsilon}.$$

Lemma 3.6. *Für ein solches s gilt $(1 + \epsilon) \cdot t < \frac{s^2}{4s+16}$.*

Beweis.

$$\begin{aligned}
\frac{s^2}{4s + 16} &= \frac{(12 + 24(1 + \frac{\epsilon}{3}) \cdot t)^2}{\epsilon^2 \cdot (4 \cdot \frac{12 + 24(1 + \frac{\epsilon}{3}) \cdot t}{\epsilon} + 16)} \\
&= \frac{(12 + 24(1 + \frac{\epsilon}{3}) \cdot t)^2}{\epsilon \cdot 4 \cdot (12 + 24(1 + \frac{\epsilon}{3}) \cdot t) + \epsilon^2 \cdot 16}
\end{aligned}$$

$$
\begin{aligned}
&> \frac{3 \cdot (12 + 24(1+\frac{\epsilon}{3}) \cdot t)^2}{4 \cdot (12 + 24(1+\frac{\epsilon}{3}) \cdot t) + \frac{16}{3}} && (\epsilon < \frac{1}{3}) \\
&> \frac{3 \cdot (12 + 24(1+\frac{\epsilon}{3}) \cdot t)^2}{5 \cdot (12 + 24(1+\frac{\epsilon}{3}) \cdot t)} && (12 + 24(1+\frac{\epsilon}{3}) \cdot t > \frac{16}{3}) \\
&> \frac{3}{5} \cdot (12 + 24 \cdot t) && (\epsilon > 0) \\
&> (1 + \frac{1}{3}) \cdot t \quad > \quad (1+\epsilon) \cdot t
\end{aligned}
$$

□

Insbesondere gilt also $(1+\frac{\epsilon}{3}) \cdot t < \frac{s^2}{4s+16}$ und $t < \frac{s^2}{4s+16}$.

Lemma 3.7. *Seien $p, p', q, q' \in P$ wie in Lemma 3.5. Dann gilt*

(1) Ist (p,q) t-distanzerhaltend, dann ist (p',q') $(1+\frac{\epsilon}{3})t$-distanzerhaltend.

(2) Ist (p,q) $(1+\frac{\epsilon}{3})t$-distanzerhaltend, dann ist (p',q') $(1+\epsilon)t$-distanzerhaltend.

Beweis. Zu 1. Sei (p,q) t-distanzerhaltend. Da nach Lemma 3.6 $1 \leq t < \frac{s^2}{4s+16}$ gilt, sind die Voraussetzungen von Lemma 3.5 erfüllt. Also ist (p',q') t'-distanzerhaltend, wobei t' in selbigem Lemma gegeben ist. $t < \frac{s^2}{4s+16}$ ist äquivalent zu $0 < s^2 - 4st - 16t$. Daraus folgt, dass $s \geq 4t \geq 4$ ist. Es ergibt sich damit und durch unsere spezielle Wahl von s

$$
t' = \frac{(1+\frac{4}{s}) \cdot t}{1 - 4(1+\frac{4}{s}) \cdot \frac{t}{s}} \overset{s \geq 4}{\leq} \frac{(1+\frac{4}{s})t}{1 - 4(1+\frac{4}{4}) \cdot \frac{t}{s}} \leq \frac{(1+\frac{4}{s})t}{1 - 8\frac{t}{s}} \overset{\text{Einsetzen}}{=} (1+\frac{\epsilon}{3})t.
$$

Zu 2. Sei (p,q) $(1+\frac{\epsilon}{3})t$-distanzerhaltend. Nach Lemma 3.6 gilt $(1+\frac{\epsilon}{3})t \leq \frac{s^2}{4s+16}$ und wir können auch hier wieder Lemma 3.5 anwenden. Dieses besagt dann, dass (p',q') t''-distanzerhaltend ist, wobei sich t'' ergibt als

$$
t'' = \frac{(1+\frac{4}{s}) \cdot (1+\frac{\epsilon}{3})t}{1 - 4(1+\frac{4}{s}) \cdot (1+\frac{\epsilon}{3})\frac{t}{s}}.
$$

Da $0 < \epsilon < \frac{1}{3}$, ist

$$
s = \frac{12 + 24(1+\frac{\epsilon}{3}) \cdot t}{\epsilon} \geq \frac{4(1+\frac{\epsilon}{3})t}{\epsilon} \geq \frac{4(1+\frac{\epsilon}{3})}{\epsilon} \geq \frac{4(1+\frac{\epsilon}{3})}{1-\frac{\epsilon}{3}}.
$$

Desweiteren ist

$$
\begin{aligned}
& s \;\geq\; \frac{4(1+\frac{\epsilon}{3})}{1-\frac{\epsilon}{3}} \quad \Leftrightarrow \quad s \cdot (2 - (1+\frac{\epsilon}{3})) \;\geq\; 4 \cdot (1+\frac{\epsilon}{3}) \\
\Leftrightarrow \quad & 2s \;\geq\; 4 \cdot (1+\frac{\epsilon}{3}) + s \cdot (1+\frac{\epsilon}{3}) \quad \Leftrightarrow \quad 2 \;\geq\; (1+\frac{4}{s}) \cdot (1+\frac{\epsilon}{3}).
\end{aligned}
$$

Also ist

$$
t'' \;\leq\; \frac{(1+\frac{4}{s}) \cdot (1+\frac{\epsilon}{3})t}{1 - 4 \cdot 2 \cdot \frac{t}{s}} \;=\; \frac{(1+\frac{4}{s}) \cdot (1+\frac{\epsilon}{3})t}{1 - 8\frac{t}{s}} \overset{\text{Einsetzen}}{=} (1+\frac{\epsilon}{3})^2 \cdot t \;\leq\; (1+\epsilon) \cdot t.
$$

□

Seien nun für alle $1 \leq i \leq m$ a_i und b_i zwei beliebige, aber feste Elemente aus A_i und B_i. Seien weiter α_i und β_i die Elemente von P, für die $a_i = \delta(p_1, \alpha_i)$ und $b_i = \delta(p_1, \beta_i)$ gilt.

Sind nun $x \in A_i$ und $y \in B_i$ und $x = \delta(p_1, p)$ und $y = \delta(p_1, q)$, so besagt Lemma 3.7 insbesondere, dass falls die Kante (p,q) t-distanzerhaltend ist, auch (α_i, β_i) $(1+\frac{\epsilon}{3})t$-distanzerhaltend ist.

Der Einfachheit halber nennen wir die Tupel (A_i, B_i) (t, ϵ)-*distanzerhaltend*, falls (α_i, β_i) $(1 + \frac{\epsilon}{3})t$-distanzerhaltend ist.

Als Nächstes konstruieren wir aus der WSPD einen gerichteten Graphen H und zeigen dann, dass jede t-distanzerhaltende Approximation von P einem Pfad in H entspricht. Die Knoten von H sind genau die $2m$ Mengen A_i und B_i, und die Kanten sind wie folgt definiert:

(1) Für alle $1 \leq i \leq m$ ist (A_i, B_i) genau dann eine Kante, wenn (A_i, B_i) (t, ϵ)-distanzerhaltend ist und $x_n \in B_i$.

(2) Für alle $1 \leq i < j \leq m$ ist (A_i, A_j) genau dann eine Kante, wenn (A_i, B_i) (t, ϵ)-distanzerhaltend ist und $A_j \cap B_i \neq \emptyset$.

Satz 3.8. *Jede t-distanzerhaltende Approximation $Q = (q_1, q_2, \ldots, q_k)$ von P entspricht einem Pfad R der Länge k in H von einer Menge A_i, die x_1 enthält, zu einer Menge B_j, die x_n enthält.*

Beweis. Wir werden nun induktiv Q zu einem Pfad R in H konvertieren.

Sei y_i das Element der Menge S, für das $y_i = \delta(p_1, q_i)$ gilt. Da nach Bedingung ja $q_1 = p_1$ gilt, ist also auch $y_1 = x_1$. Sei weiter i_1 ein solcher Index, für den $y_1 \in A_{i_1}$ und $y_2 \in B_{i_1}$ gilt (dieser existiert nach Definition der WSPD). Dann wählen wir A_{i_1} als ersten Knoten des Pfades R.

Nehmen wir jetzt an, dass wir bereits für ein l mit $1 \leq l < k - 1$ den Kantenzug $(q_1, \ldots, q_l)$ zu dem Teilpfad $(A_{i_1}, \ldots, A_{i_l})$ von R umgewandelt haben, sodass $y_l \in A_{i_l}$ und $y_{l+1} \in B_{i_l}$. Wir wählen jetzt i_{l+1} als den Index, für den $y_{l+1} \in A_{i_{l+1}}$ und $y_{l+2} \in B_{i_{l+1}}$ ist. Solch ein Index existiert nach Definition der WSPD und ist eindeutig. Nach Annahme ist (q_l, q_{l+1}) t-distanzerhaltend und $y_l \in A_{i_l}$ und $y_{l+1} \in B_{i_l}$. Wie oben beobachtet folgt dann aus Lemma 3.7 (1), dass das Tupel (A_i, B_i) (t, ϵ)-distanzerhaltend ist. Außerdem ist der Schnitt von $A_{i_{l+1}}$ mit B_{i_l} nicht leer, da y_{l+1} in beiden Mengen liegt. Diese beiden Eigenschaften sind hinreichend dafür, dass $(A_{i_l}, A_{i_{l+1}})$ eine Kante in H ist. Deshalb können wir $A_{i_{l+1}}$ als nächsten Knoten von R wählen. Wir haben damit also $(q_1, \ldots, q_l, q_{l+1})$ zu dem Pfad $(A_{i_1}, \ldots, A_{i_l}, A_{i_{l+1}})$ in H umgewandelt, sodass $y_{l+1} \in A_{i_{l+1}}$ und $y_{l+2} \in B_{i_{l+1}}$.

Nehmen wir an, dass wir bereits $(q_1, q_2, \ldots, q_{k-1})$ zu dem Pfad $(A_{i_1}, \ldots, A_{i_{k-1}})$ umgewandelt haben, wobei $y_{k-1} \in A_{i_{k-1}}$ und $y_k \in B_{i_{k-1}}$. Nach Annahme ist $q_k = p_n$ und somit ist auch $y_k = x_n$. Insbesondere ist also x_n ein Element von $B_{i_{k-1}}$. Da die Kante (q_{k-1}, q_k) t-distanzerhaltend ist, ist wieder wegen Lemma 3.7 (1) $(A_{i_{k-1}}, B_{i_{k-1}})$ (t, ϵ)-distanzerhaltend. Also ist $(A_{i_{k-1}}, B_{i_{k-1}})$ eine Kante in H. Wir fügen $B_{i_{k-1}}$ zum Pfad hinzu, und erhalten als Gesamtergebnis $R = (A_{i_1}, \ldots, A_{i_{k-1}}, B_{i_{k-1}})$. □

Wir haben also gezeigt, dass jede t-distanzerhaltende Approximation von P einem Pfad in H entspricht, der dieselbe Zahl von Knoten hat. Der nächste Satz zeigt, dass dies auch umgekehrt der Fall ist, unter der Einschränkung, dass die Approximation um einen kleinen Teil vom gewünschten t-Wert abweichen darf.

Satz 3.9. *Jeder Pfad $R = (A_{i_1}, \ldots, A_{i_{k-1}}, B_{i_{k-1}})$ in H mit $x_1 \in A_{i_1}$ und $x_n \in B_{i_{k-1}}$ entspricht einer $(1 + \epsilon)t$-distanzerhaltenden Approximation Q von P, die k Knoten besitzt.*

Für den Beweis wählen wir eine ähnliche Vorgehensweise wie beim Beweis von Satz 3.8: Wir konvertieren den Pfad R induktiv zu einer Approximation von P und zeigen anschließend, dass diese auch wirklich $(1 + \epsilon)t$-distanzerhaltend ist.

Beweis. Sei wieder y_i das Element der Menge S, für das $y_i = \delta(p_1, q_i)$ gilt. Da x_1 in A_{i_1} liegt, können wir als ersten Knoten von Q $q_1 = p_1$ wählen.

Nehmen wir an, dass wir bereits für ein l mit $1 \leq l < k-1$ den Teilpfad $(A_{i_1}, \ldots, A_{i_l})$ in den Kantenzug $(q_1, \ldots, q_l)$ umgewandelt haben, sodass y_1 $(= x_1) \in A_{i_1}$ und für alle $1 < j \leq l$ $y_j \in A_{i_j} \cap B_{i_{j-1}}$. Betrachten wir jetzt die Kante $(A_{i_l}, A_{i_{l+1}})$, gibt es ein $y \in A_{i_{l+1}} \cap B_{i_l}$, da der Schnitt nach Definition der Kanten von H nicht leer sein kann. Dann wählen wir als q_{l+1} den Knoten γ von P, für den $y = \delta(p_1, \gamma)$ gilt und erhalten $(q_1, \ldots, q_l, q_{l+1})$.

Nehmen wir an, dass wir bereits $(A_{i_1}, \ldots, A_{i_k-1})$ zu $(q_1, \ldots, q_{k-1})$ konvertiert haben. Nach Voraussetzung ist $x_n \in B_{k-1}$. Wir wählen dann $q_k = p_n$ und fügen q_k zum Kantenzug Q hinzu. Insgesamt haben wir also gezeigt, wie man R zu einem Kantenzug $Q = (q_1, q_1, \ldots, q_k)$ mit gleich vielen Knoten umwandeln kann.

Jetzt bleibt zu zeigen, dass Q auch tatsächlich $(1+\epsilon)t$-distanzerhaltend ist. Dazu betrachten wir ein j mit $1 \leq j < k$. Nach unserer Konstruktion von Q ist $y_j \in A_{i_j}$ und $y_{j+1} \in B_{i_j}$. Nach Voraussetzung ist die Kante (A_{i_j}, B_{i_j}) zudem (t, ϵ)-distanzerhaltend. Aus Lemma 3.7 (2) folgt dann, dass alle Kanten (α, β) mit $\delta(p_1, \alpha) \in A_{i_j}$ und $\delta(p_1, \beta) \in B_{i_{j+1}}$ $(1+\epsilon)t$-distanzerhaltend sind, insbesondere ist also die Kante (q_j, q_{j+1}) zwischen den zu y_j und y_{j+1} gehörigen Knoten $(1+\epsilon)t$-distanzerhaltend. Diese Eigenschaft gilt für alle $1 \leq j < k$, woraus folgt, dass der von uns konstruierte Kantenzug Q eine $(1+\epsilon)t$-distanzerhaltende Approximation von P ist. □

Durch diese beiden Sätze wird schnell klar, was wir tun müssen, um eine $(1+\epsilon)t$-distanzerhaltende Approximation eines Pfades P zu erhalten: Wir konstruieren einfach den Graphen H und führen darin eine Breitensuche aus, um einen kürzesten Pfad von einer Menge A_{i_1}, die x_1 enthält, zu einer Menge B_{i_k}, die x_n enthält, zu bestimmen. Jetzt zeigen wir noch, dass dieses Vorgehen auch tatsächlich eine bis auf ein ϵ genaue Lösung des MVPS-Problems liefert.

Satz 3.10. *Sei κ_t die Zahl von Knoten, aus der eine minimale t-distanzerhaltende Approximation von P besteht. Dann entspricht ein kürzester Pfad R in H von einer Menge A_{i_1}, die x_1 enthält, zu einer Menge B_{i_k}, die x_n enthält, einer $(1+\epsilon)t$-distanzerhaltenden Approximation von P, die aus maximal κ_t Knoten besteht.*

Beweis. Sei Q eine minimale t-distanzerhaltende Approximation von P und κ_t ihre Knotenzahl. Nach Satz 3.8 entspricht Q einem Pfad in H von einer Menge A_{i_1}, die x_1 enthält, zu einer Menge B_{i_k}, die x_n enthält. Dieser Pfad besteht aus κ_t Knoten. Folglich hat R maximal κ_t Knoten und nach Satz 3.9 entspricht R einer $(1+\epsilon)t$-distanzerhaltenden Approximation von P, die maximal κ_t Knoten hat. □

Jetzt haben wir aber noch ein Problem: H hat $2m = O(sn)$ Knoten und somit unter Umständen $\Theta(s^2n^2)$ Kanten. Das können wir uns aber nicht erlauben, wenn wir eine bessere Laufzeit als die des exakten Algorithmus anstreben.

Im folgenden Abschnitt werden wir sehen, wie man einen kürzesten Pfad in H finden kann, ohne H vollständig zu konstruieren.

Der Algorithmus

Der Algorithmus besteht aus fünf Teilen, von denen wir die ersten drei im Prinzip schon betrachtet haben. Sei $P = (p_1, p_2, \ldots, p_n)$ ein Kantenzug. Die ersten drei Teile sind folgende:

(1) Berechne $S = (x_1, x_2, \ldots, x_n)$, wobei $x_i = \delta(p_1, p_i)$ für alle $1 \leq i \leq n$.

(2) Berechne aus S den Split-Tree T und daraus eine WSPD $\{(A_i, B_i)\}_{1 \leq i \leq m'}$ mit der Trennungsrate $s = \frac{12+24(1+\frac{\epsilon}{3})t}{\epsilon}$. Nehme wieder o.B.d.A. an, dass für alle $1 \leq i \leq m'$ alle Elemente aus A_i kleiner sind als alle aus B_i.

(3) Seien $a_i \in A_i$ und $b_i \in B_i$ für alle $1 \leq i \leq m'$ feste Elemente und seien α_i und β_i die Knoten von P, für die $a_i = \delta(p_1, \alpha_i)$ und $b_i = \delta(p_1, \beta_i)$. Falls (α_i, β_i) nicht $(1 + \frac{\epsilon}{3})t$-distanzerhaltend ist, verwirf das korrespondierende Tupel (A_i, B_i), ansonsten behalte es.

Der Einfachheit halber beschreiben wir die „ausgedünnte“ WSPD durch $\{(A_i, B_i)\}_{1 \leq i \leq m}$, wobei m die Zahl der verbleibenden Tupel ist. Bevor wir zu Schritt (4) kommen, zeigen wir zunächst, wie man mit Hilfe einer Breitensuche im Split-Tree T einen kürzesten Pfad in H bestimmen kann.

Für alle $1 \leq i \leq m$ sei u_i der Knoten des Split-Trees T, der A_i repräsentiert, und v_i derjenige, der B_i repräsentiert. Wir nennen die u_i auch *A-Knoten* und die v_i *B-Knoten*. Es ist nicht ausgeschlossen, dass ein Knoten von T mehrere A_i und B_i repräsentiert. Folglich speichert jeder Knoten eine Liste seiner A_i und B_i.

Die Breitensuche die wir verwenden, zeigt Ähnlichkeiten zu der, die im Informatik III Skript [5] erklärt wird. Für jeden Knoten v des Split-Trees T speichern wir dabei drei Variablen:

- $color[v]$, die einen Wert aus $\{white, gray, black\}$ hat.
- $dist[v]$, die die aktuelle Distanz von einem Knoten A_i in H, der x_1 enthält, zum Knoten v speichert.
- $parent[v]$, die den Vater von w im BFS-Wald speichert.

Sind die Knoten von T mit den Zahlen $1, 2, \ldots, n'$ benannt sind, können wir *color*, *dist* und *parent* zum Beispiel durch drei Arrays der Größe n' realisieren. Die Breitensuche sieht dann so aus:

Schritt 1: Für alle Knoten k von T, setze $color[k] = white$, $dist[k] = \infty$ und $parent[k] = null$.

Schritt 2: Initialisiere eine leere Warteschlange W (z.B. als eine verkettete Liste). Starte bei dem Blatt, das x_1 speichert (dem „linkesten“ Blatt) und laufe im Baum aufwärts bis zur Wurzel. Für alle besichtigten Knoten k, tue Folgendes:

Setze $color[k] = gray$. Falls k ein A-Knoten ist, setze $dist[k] = 0$. Füge k in W ein.

Schritt 3: Entferne das erste Element k von W. Setze $color[k] = black$. Für alle i mit $u_i = k$ (also für alle mit k gespeicherten Mengen A_i) tue Folgendes:

Falls $x_n \in B_i$, setze $dist[v_i] = dist[k] + 1$, $parent[v_i] = k$ und $z = v_i$ und gehe zu Schritt 4.
Falls $x_n \notin B_i$ und $color[v_i] == white$, führe die Schritte 3.1 und 3.2 aus.

Schritt 3.1: Starte bei v_i und laufe im Baum aufwärts bis zum ersten nicht weißen Knoten. Für alle besichtigten Knoten k', tue Folgendes:

Setze $color[k'] = gray$. Falls k' ein A-Knoten ist, setze $dist[k'] = dist[k] + 1$ und $parent[k'] = k$ und füge k' in W ein.

Schritt 3.2: Starte bei v_i und besuche alle Knoten des Teilbaums von T, dessen Wurzel v_i ist. Für alle besichtigten Knoten k', tue Folgendes:

Setze $color[k'] = gray$. Falls k' ein A-Knoten ist, setze $dist[k'] = dist[k]+1$, $parent[k'] = k$ und füge k' in W ein.

Sind alle Knoten k' besichtigt worden, gehen zurück zu Schritt 3.

Schritt 4: Berechne den Pfad $Z = (z, parent[z], parent[parent[z]], parent^3[z], \ldots, parent^{k-1}[z])$, wobei $k = dist[z]+1$. Gib den umgekehrten Pfad $Z' = (parent^{k-1}[z], \ldots, , parent[z], z)$ zurück.

Dass diese modifizierte Breitensuche einen kürzesten Pfad in H zurückgibt, ist nicht sofort erkennbar. Auf einen vollständigen Beweis wollen wir an dieser Stelle zwar verzichten, aber die wichtigsten Punkte skizzieren.

Wir können beobachten, dass, falls ein B-Knoten v_i weiß ist, auch alle Knoten im Unterbaum von v_i weiß sind. Ist jedoch v_i nicht-weiß, wurden auch alle Knoten im Unterbaum von v_i schon von einem anderen Knoten aus in Schritt 3.2 betrachtet und sind deshalb ebenfalls nicht-weiß. In diesem Fall werden auch die Schritte 3.1 und 3.2 für v_i *nicht* ausgeführt. Ist v der erste nicht-weiße Knoten, der in Schritt 3.1 erreicht wird, sind alle Knoten auf dem Pfad von v zur Wurzel des Split-Trees nicht-weiß. Aus diesen Beobachtungen folgt, dass im Laufe der Breitensuche jede Kante des Baumes maximal einmal betrachtet wird.

Ist k das erste Element der Warteschlange, so hat es die momentan geringste Distanz zu einer Menge A_i, die x_1 enthält. Sei A_l eine Menge, für die $u_l = k$ gilt. Wird jetzt u_l bearbeitet, werden alle anderen von A_l aus (in H) erreichbaren Mengen A_j betrachtet. Diese Mengen sind genau die, die einen nicht-leeren Schnitt mit B_l haben (deshalb betrachten wir in Schritt 3 auch v_l). Für jedes A_j wird dann dessen Distanz zu A_i verringert (nämlich von ∞ auf $d[k] + 1$).

Ist jedoch x_n im zu A_l gehörigen B_l bereits enthalten, ist die Breitensuche beendet, da dann ein kürzester Weg von A_i zu B_l bestimmt wurde.

Ein genauer, vollständiger Beweis kann unter Verwendung der obigen Beobachtungen analog zum Beweis der Breitensuche im Informatik III Skript [5] oder zu dem in Cormen et al. [3] durchgeführt werden. Ist m die Kantenzahl in T und z die Knotenzahl, ergibt sich dabei insbesondere die für eine Breitensuche übliche Laufzeit $O(m + z)$. Aber $O(m + z) = O(sn) = O(\frac{t}{\epsilon}n)$. Darum erhalten wir für die Laufzeit der Breitensuche insgesamt die Komplexitätsklasse $O(\frac{t}{\epsilon}n)$.

Die letzten beiden Teile des Algorithmus sind dann folgende:

(4) Führe die oben aufgeführte modifizierte Breitensuche in T durch, um einen kürzesten Pfad Z' von einer Menge A_{i_1}, die x_1 enthält, zu einer Menge $B_{i_{k-1}}$, die x_n enthält, zu bestimmen.

(5) Konvertiere Z', der den Kriterien von Satz 3.9 entspricht, wie im Beweis desselben Satzes zu einer Approximation von P.

Abschließend betrachten wir noch die Laufzeit des Algorithmus. S zu berechnen kostet uns $O(n)$ Zeit. Nach Satz 3.4 können wir Teil 2 in $O(n \log n + sn)$ Zeit schaffen, und da $O(sn) = O(\frac{t}{\epsilon} \cdot n)$, dauert dieser Teil also $O(n \log n + \frac{t}{\epsilon}n)$ Zeit. Somit kostet auch Teil 3 $O(\frac{t}{\epsilon}n)$ Zeit, da wir jedes Tupel der WSPD einmal betrachten. Wie oben gesehen, weist die Breitensuche dieselbe asymptotische Laufzeit auf.

Nun bleibt noch der letzte Teil des Algorithmus. Um den Pfad in H zu konvertieren, wählen wir, wie gezeigt, p_1 als ersten und p_n als letzten Knoten. Einer der wichtigen Schritte für die restlichen Knoten besteht darin, ein Element aus $A_{i_{l+1}} \cap B_{i_l}$ auszuwählen. Da wir wissen, dass der Schnitt nicht leer ist und $A_{i_{l+1}}$ und B_{i_l} durch Knoten des Split-Trees repräsentiert werden (nämlich $u_{i_{l+1}}$ und v_{i_l}), die wiederum die Blätter in ihrem jeweiligen Unterbaum zusammenfassen, muss entweder $A_{i_{l+1}} \subseteq B_{i_l}$ sein oder umgekehrt. Also wählen wir zum Beispiel das Minimum μ des Intervalls, das mit $u_{i_{l+1}}$ gespeichert ist, und überprüfen, ob es im Intervall von v_{i_l} liegt. Ist das der Fall, liegt dieses im Schnitt und wir wählen den zu μ gehörenden Knoten als Nächsten. Sonst ist $B_{i_l} \subset A_{i_{l+1}}$ und wir können das Minimum des Intervalls von v_{i_l} wählen. Dieses Vorgehen dauert für jede Kante $(A_{i_l}, A_{i_{l+1}})$ jeweils konstante Zeit. Insgesamt kostet Teil 5 des Algorithmus also $O(\frac{t}{\epsilon}n)$ Zeit. Zusammenfassend können wir festhalten:

Satz 3.11. *Sei $P = (p_1, p_2, \ldots, p_n)$ ein Kantenzug in $\mathbb{R}^d$, sei $t \geq 1$ und $0 < \epsilon < \frac{1}{3}$ und sei κ die Knotenzahl der minimalen t-distanzerhaltenden Approximationen von P. Dann können wir in $O(n \log n + \frac{t}{\epsilon} n)$ eine $(1+\epsilon)t$-distanzerhaltende Approximation Q von P mit maximal κ Knoten berechnen.*

3.3 Algorithmus für das MDPS-Problem

Ähnlich wie in Kapitel 2 werden wir jetzt versuchen, mit Hilfe des eben vorgestellten approximativen Algorithmus für das MVPS-Problem eine Lösung des MDPS-Problems zu konstruieren.

Sei dazu k die Knotenzahl und $P = (p_1, p_2, \ldots, p_n)$ der Kantenzug, für die wir das MDPS-Problem lösen wollen. Wie in Kapitel 2 sei $t^* = \min\{t \geq 1 \mid \kappa_t \leq k\}$ die Lösung des Problems.

Lemma 3.12. *Eine t^*-distanzerhaltende Approximation Q von P mit höchstens k Knoten muss eine Kante (p, q) besitzen, die eine Abweichung von t^* hat.*

Beweis. Gäbe es keine solche Kante, gäbe es also ein $t < t^*$, sodass Q auch t-distanzerhaltend ist. Dann ist Q aber eine t-distanzerhaltende Approximation von P mit höchstens k Knoten – ein Widerspruch zur Wahl von t^*. □

Der Algorithmus zur approximativen Lösung des MDPS-Problems basiert im Wesentlichen auf dem folgenden Lemma.

Lemma 3.13. *Sei $t \geq 1$, $0 < \epsilon < \frac{1}{3}$ und sei $Q = (q_1, q_2, \ldots, q_k)$ eine $(1+\epsilon)t$-distanzerhaltende Approximation, die durch den Algorithmus von Satz 3.11 entstanden ist. Ist k' die Anzahl der Knoten von Q, dann gilt:*

(1) Falls $k' \leq k$, dann ist $t^* \leq (1+\epsilon) \cdot t$.

(2) Falls $k' > k$, dann ist $t^* > t$.

Beweis. Zu 1. Sei $k' \leq k$. Sei t' die exakte Abweichung des Kantenzugs Q von P, also

$$t' = \max\{\frac{\delta(q_i, q_{i+1})}{|q_i q_{i+1}|} \mid 1 \leq i < k\}.$$

Nach Voraussetzung gilt auch $t' \leq (1+\epsilon) \cdot t$. Aus Lemma 2.2 folgt damit, dass $\kappa_{(1+\epsilon)t} \leq \kappa_{t'}$. Da $\kappa_{t'}$ aber die Knotenzahl einer minimalen t'-distanzerhaltenden Approximation ist, muss $\kappa_{t'} \leq k'$ gelten. Zusammen mit der Annahme folgt dann $\kappa_{(1+\epsilon)t} \leq k$. Wegen Lemma 2.2 muss dann aber $t^* \leq (1+\epsilon) \cdot t$ gelten.

Zu 2. Sei nun $k' > k$. Nach Satz 3.11 gilt $k' \leq \kappa_t$. Also ist auch $k < \kappa_t$. Wie oben folgt nun aus Lemma 2.2, dass $t^* > t$ sein muss, da für t^* ja $\kappa_{t^*} \leq k$ gilt. □

Eine erste Approximation von t^*

Sei nun $0 < \epsilon < \frac{1}{3}$ fest. Der folgende Algorithmus berechnet eine Annäherung τ von t^* bis auf einen Faktor von 2.

Schritt 1: Setze $\tau = 2$.

Schritt 2: Führe den Algorithmus von Satz 3.11 aus. Sei Q die dadurch erhaltende Approximation und k' deren Knotenzahl.

Schritt 3: Falls $k' > k$, setze $\tau = 2 \cdot \tau$ und gehe zu Schritt 2. Falls $k' \leq k$, gib τ zurück.

Dass nach der Terminierung des Algorithmus

$$\frac{\tau}{2} < t^* \leq (1+\epsilon) \cdot \tau \qquad (*)$$

gilt, folgt direkt aus Lemma 3.13. Betrachten wir nun noch die Laufzeit des Algorithmus. Der im vorangehenden Kapitel vorgestellte Algorithmus zur Lösung des MVPS-Problems berechnet zunächst einen Split-Tree T und erstellt dann daraus zur gegebenen Trennungsrate s eine WSPD. Da T aber nur vom Eingabe-Kantenzug P abhängt, genügt es, T nur einmal zu berechnen und dann daraus die verschiedenen WSPDs zu erstellen. Bei gegebenem T dauert die Lösung des MVPS-Problems für t und ϵ mit Satz 3.11 dann nur noch $O(\frac{t}{\epsilon} \cdot n)$. Im obigen Algorithmus wird Schritt 2 $O(\log \tau)$ mal ausgeführt, da τ ja bei jeder Iteration verdoppelt wird. Die Laufzeit für die Approximation von t^* bei gegebenem Split-Tree T ist damit die Folgende:

$$O(\sum_{i=1}^{\log_2 \tau} (\frac{2^i}{\epsilon} \cdot n)) = O(\frac{\tau}{\epsilon} \cdot n) = O(\frac{t^*}{\epsilon} \cdot n).$$

Eine bessere Approximation von t^* mit Hilfe von binärer Suche

Bevor wir uns einem Algorithmus für eine bessere Approximation von t^* als oben zuwenden, müssen wir zunächst ein dafür essentielles Lemma beweisen.

Sei $S = (x_1, x_2, \ldots, x_n)$ wie in Kapitel 3.2 und $\{(A_i, B_i)\}_{1 \leq i \leq m}$ eine WSPD mit einer Trennungsrate von

$$s = \frac{4 + 8(1+\epsilon)^3 \cdot \tau}{\epsilon}.$$

Seien für alle $1 \leq i \leq m$ a_i und b_i feste Elemente aus A_i bzw. B_i und α_i und β_i die Knoten von P, für die $a_i = \delta(p_1, \alpha_i)$ und $b_i = \delta(p_1, \beta_i)$ ist. Sei $t_i = \frac{\delta(\alpha_i, \beta_i)}{|\alpha_i \beta_i|}$. Dann gilt:

Lemma 3.14. *Es gibt ein j mit $1 \leq j \leq m$ und*

$$\frac{t_j}{1+\epsilon} \leq t^* \leq (1+\epsilon) \cdot t_j$$

Beweis. Nach Lemma 3.12 gibt es zwei Knoten p und q, für die $t^* = \frac{\delta(p,q)}{|pq|}$ gilt. Sei j der Index, sodass $\delta(p_1, p) \in A_j$ und $\delta(p_1, q) \in B_j$. Solch ein Index existiert nach Definition der WSPD. Analog zum Beweis von Lemma 3.6 kann man errechnen, dass $t^* < \frac{s^2}{4s+16}$ ist. Somit ist $s^2 > 4st^* + 16t^*$, insbesondere also $s > 4 \cdot t^* \geq 4$. Durch Anwendung von Lemma 3.5 folgt dann, dass (α_j, β_j) t'-distanzerhaltend ist, wobei

$$t' = \frac{(1 + \frac{4}{s}) \cdot t^*}{1 - 4(1 + \frac{4}{s})\frac{t^*}{s}}.$$

Da wie gezeigt $s \geq 4$, ist

$$t' \leq \frac{(1 + \frac{4}{s}) \cdot t^*}{1 - 4 \cdot (1 + \frac{4}{4})\frac{t^*}{s}} \leq \frac{(1 + \frac{4}{s}) \cdot t^*}{1 - 8\frac{t^*}{s}}. \qquad (1)$$

Aus $(*)$ folgt, dass

$$s = \frac{4 + 8(1+\epsilon)^3 \cdot \tau}{\epsilon} \quad \geq \quad \frac{4 + 8(1+\epsilon)t^*}{\epsilon}.$$

Durch Umformen erhält man daraus

$$\frac{(1 + \frac{4}{s}) \cdot t^*}{1 - 8 \cdot \frac{t^*}{s}} \quad \leq \quad (1+\epsilon) \cdot t^*. \qquad (2)$$

Insgesamt ist damit

$$t_j = \frac{\delta(\alpha_j, \beta_j)}{|\alpha_j \beta_j|} \leq t' \overset{(1)}{\leq} \frac{(1+\frac{4}{s}) \cdot t^*}{1 - 8\frac{t^*}{s}} \overset{(2)}{\leq} (1+\epsilon) \cdot t^*.$$

Somit haben wir die erste Ungleichung bewiesen.

Nun zur zweiten. Wir wissen schon, dass $t_j \leq (1+\epsilon) \cdot t^*$ ist. Auch hier kann man analog zum Beweis von Lemma 3.6 errechnen, dass $4st_j + 16t_j < s^2$ ist. Damit ist wieder die Bedingung aus Lemma 3.5 erfüllt. Da die Kante (α_j, β_j) t_j-distanzerhaltend ist, folgt aus selbigem Lemma, dass (p, q) t''-distanzerhaltend, wobei sich t'' ergibt als

$$t'' = \frac{(1+\frac{4}{s}) \cdot t_j}{1 - 4(1+\frac{4}{s}) \cdot \frac{t_j}{s}}.$$

Wie oben ist $s \geq 4$, sodass sich der Ausdruck zu

$$t'' \leq \frac{(1+\frac{4}{s}) \cdot t_j}{1 - 4(1+\frac{4}{4}) \cdot \frac{t_j}{s}} = \frac{(1+\frac{4}{s}) \cdot t_j}{1 - 8 \cdot \frac{t_j}{s}} \tag{3}$$

vereinfacht. Aus dem ersten Teil des Beweises und aus $(*)$ folgt $t_j \leq (1+\epsilon) \cdot t^* \leq (1+\epsilon)^2 \tau$. Somit ist

$$s = \frac{4 + 8(1+\epsilon)^3 \cdot \tau}{\epsilon} \geq \frac{4 + 8(1+\epsilon) \cdot t_j}{\epsilon}.$$

Dies lässt sich umformen zu

$$\frac{(1+\frac{4}{s}) \cdot t_j}{1 - 8 \cdot \frac{t_j}{s}} \leq (1+\epsilon) \cdot t_j. \tag{4}$$

Insgesamt erhalten wir damit

$$t^* = \frac{\delta(p, q)}{|pq|} \leq t'' \overset{(3)}{\leq} \frac{(1+\frac{4}{s}) \cdot t_j}{1 - 8 \cdot \frac{t_j}{s}} \overset{(4)}{\leq} (1+\epsilon) \cdot t_j.$$

□

Nun können wir uns dem eigentlichen Algorithmus zuwenden. Dieser lautet wie folgt:

Schritt 1: Berechne τ wie oben als Approximation von t^* bis auf einen Faktor von 2

Schritt 2: Berechne $S = (x_1, x_2, \ldots, x_n)$ (wobei $x_i = \delta(p_1, p_i)$). Erstelle daraus den Split-Tree T und eine WSPD $\{(A_i, B_i)\}_{1 \leq i \leq m}$ mit Trennungsrate $s = \frac{4+8(1+\epsilon)^3 \cdot \tau}{\epsilon}$. Wähle für alle $1 \leq i \leq m$ a_i und b_i als feste Elemente aus A_i bzw. B_i und α_i und β_i als die Elemente von P, für die $a_i = \delta(p_1, \alpha_i)$ und $b_i = \delta(p_1, \beta_i)$ gilt.

Schritt 3: Setze für $1 \leq i \leq m$ $t_i \coloneqq \frac{\delta(\alpha_i, \beta_i)}{|\alpha_i \beta_i|}$ und erstelle aus den t_i eine Liste. Setze $t_0 \coloneqq 1$ und sortiere die t_i für $0 \leq i \leq m$.

Schritt 4: Entferne alle Duplikate und die Werte aus der Liste, die größer als $(1+\epsilon)^2 \tau$ sind.

Sei L die übrigbleibende sortierte Liste der Größe m', für die $1 = t_0 < t_1 < \ldots < t_{m'} \leq (1+\epsilon)^2 \cdot \tau$ gilt.

Schritt 5: Führe eine binäre Suche nach t^* in L durch.

Dass uns eine binäre Suche in L tatsächlich einen geeigneten Wert zurückgibt, ist nicht sofort klar. Deshalb werden wir jetzt zeigen, dass die folgende Invariante während der gesamten Suche erhalten bleibt und die Gelegenheit nutzen, die Suche genauer zu beschreiben. Sei l die untere Grenze des noch zu durchsuchenden Intervalls und r die obere. Dann gilt:

Invariante 3.15. *l und r sind Zahlen mit $0 \leq l < r \leq m'$ und $t_l \leq t^* \leq (1+\epsilon) \cdot t_r$.*

Beweis. Wir unterscheiden drei Fälle:

Fall 1: $l = 0$ und $r = m'$.

Wir betrachten das j aus Lemma 3.14. Da für dieses j ja $t_j \leq (1+\epsilon) \cdot t^*$ gilt und nach $(*)$ $(1+\epsilon) \cdot t^* \leq (1+\epsilon)^2 \cdot \tau$, kommt t_j in L vor. Jetzt gilt aber auch $t_l = 1 \leq t^* \leq (1+\epsilon) \cdot t_j \leq (1+\epsilon) \cdot t_r \leq (1+\epsilon)^2 \cdot \tau$, sodass in diesem Fall die Invariante erfüllt ist.

Fall 2: $l < r - 1$.

Angenommen die Invariante gilt. Dann wählen wir $h = \lfloor \frac{l+r}{2} \rfloor$. und berechnen die zu einer $(1+\epsilon)t_h$-distanzerhaltenden Approximation gehörende Knotenzahl k'. Jetzt überprüfen wir, ob $k' \leq k$ ist und wenden Lemma 3.13 an. Ist tatsächlich $k \leq k'$, ist $t^* \leq (1+\epsilon) \cdot t_h$. Dann gilt aber $t_l \leq t^* \leq (1+\epsilon) \cdot t_h$ und wir können $r \coloneqq h$ setzen. Sonst ist $t_h < t^* \leq (1+\epsilon) \cdot t_r$ und wir setzen $l \coloneqq h$.

Fall 3: $l = r - 1$.

Dann gilt $t_l \leq t^* \leq (1+\epsilon) \cdot t_{l+1}$. Auch hier berechnen wir zunächst die zu einer $(1+\epsilon)t_l$-distanzerhaltenden Approximation gehörende Knotenzahl k' und überprüfen, ob $k' \leq k$. Ist dies der Fall, folgt aus Lemma 3.13, dass $t^* \leq (1+\epsilon)^2 \cdot t_l$ ist. Somit gilt $t_l \leq t^* \leq (1+\epsilon)^2 \cdot t_l$ und wir geben t_l zurück.

Sonst ist $t^* > (1+\epsilon) \cdot t_l$. Jetzt betrachten wir wieder den Index j aus Lemma 3.14. Für das korrespondierende t_j gilt $t_l < \frac{t^*}{1+\epsilon} \leq t_j$. Also ist $t^* \leq (1+\epsilon) \cdot t_j$ und aus der Invariante folgt $t_{l+1} \leq t_j$. Da aber auch $\frac{t_j}{1+\epsilon} \leq t^*$ ist, ergibt sich insgesamt

$$\frac{t_{l+1}}{1+\epsilon} \leq \frac{t_j}{1+\epsilon} \leq t^* \leq (1+\epsilon) \cdot t_{l+1}$$

und wir geben $\frac{t_{l+1}}{1+\epsilon}$ zurück.

□

Invariante 3.15 bestätigt uns also, dass uns die binäre Suche in L einen solchen Wert t zurückliefert, für den $t \leq t^* \leq (1+\epsilon)^2 \cdot t$ gilt.

Betrachten wir nun noch die Laufzeit des Algorithmus. Zunächst berechnen wir zu einem gegebenen Kantenzug P das Array S in $O(n)$ Zeit, und als Nächstes daraus einen fairen Split-Tree T, was, wie wir gesehen haben, in $O(n \log n)$ Zeit möglich ist. Die Berechnung von τ kostet uns dann nur noch $O(\frac{t^*}{\epsilon} \cdot n)$ Zeit. Die WSPD, die als Nächstes bestimmt wird, hat eine Größe von $m = O(sn) = O(\frac{\tau}{\epsilon} n) = O(\frac{t^*}{\epsilon} n)$ und die Berechnung kostet dementsprechend auch $O(\frac{t^*}{\epsilon} n)$ Zeit. Da außerdem $m \leq n^2$ ist, kann das Sortieren der t_i, die wir aus der WSPD erhalten, bekanntermaßen in $O(m \log m) = O(\frac{t^*}{\epsilon} n \cdot \log n)$ Zeit geschehen. Das Aussortieren der unbrauchbaren Werte ist offensichtlich in Linearzeit in m zu bewerkstelligen.

Die binärer Suche macht $O(\log m') = O(\log m) = O(\log n)$ Iterationen. In jeder dieser Iterationen führen wir mit gegebenem Split-Tree den im Beweis von Satz 3.11 angegebenen Algorithmus mit einer Eingabegröße $t_h \leq (1+\epsilon)^2 \cdot \tau = O(t^*)$ aus. Dies führt uns zu einer Laufzeit von $O(\frac{t^*}{\epsilon} n)$ pro Durchführung, und für die gesamte binäre Suche erhalten wir $O(\frac{t^*}{\epsilon} n \cdot \log n)$ Zeit.

Nun können wir noch einen kleinen Trick anwenden: Verwenden wir im gesamten Algorithmus nicht ϵ, sonder stattdessen $\frac{\epsilon}{3}$, erhalten wir als Ergebnis ein t mit $t \leq t^* \leq (1 + \frac{\epsilon}{3})^2 \cdot t \leq (1 + \epsilon) \cdot t$. Das können wir festhalten:

Satz 3.16. *Sei $P = (p_1, p_2, \ldots, p_n)$ ein Kantenzug in $\mathbb{R}^d$, sei $2 \leq k \leq n$ und $0 < \epsilon < \frac{1}{3}$. Sei weiter t^* die kleinste Zahl t, für die eine t-distanzerhaltende Approximation von P mit maximal k Knoten existiert. Dann können wir in $O(\frac{t^*}{\epsilon} n \cdot \log n))$ eine Zahl t berechnen, für die $t \leq t^* \leq (1 + \epsilon) \cdot t$ ist.*

4 Fazit

Wir haben uns damit befasst, einen gegebenen Kantenzug P so zu approximieren, dass die Längen der Kanten der Approximation nur um einen bestimmten Faktor t von P abweichen. Im Rahmen dieser Zielsetzung haben wir zwei Probleme betrachtet: Die Minimierung der Knotenzahl der Approximation bei gegebenem t (MVPS) und die Minimierung von t bei gegebener Knotenzahl (MDPS). Dazu haben wir zunächst einen exakten Algorithmus für das MVPS-Problem betrachtet, der eine Laufzeit von $O(n^2)$ aufweist, und anschließend daraus einen exakten Algorithmus für das MDPS-Problem konstruiert, dessen Ausführung $O(n^2 \log n)$ Zeit kostet.

Die danach vorgestellten approximativen Algorithmen weisen dagegen eine deutlich bessere Laufzeit auf. Das MVPS-Problem können wir dabei bis auf ein ϵ genau in $O(n \log n + \frac{t}{\epsilon} n)$ Zeit lösen. Auch hier haben wir dieses Ergebnis verwendet, um daraus einen – diesmal approximativen – Algorithmus für das MDPS-Problem zu konstruieren. Dieser weist eine asymptotische Laufzeit von $O(\frac{t^*}{\epsilon} n \cdot \log n)$ auf, wobei t^* die exakte Lösung ist.

Die Autoren des Artikels, auf dem diese Arbeit beruht [4], zeigten 2006 in einem Experiment, dass die Laufzeit der approximativen Algorithmen nicht nur asymptotisch, sondern auch praktisch die der exakten Algorithmen deutlich unterbietet. So gelang die exakte Lösung des MVPS-Problems für einen Kantenzug mit 20.000 Punkten und einer Abweichung $t = 1.1$ in etwa 97 Sekunden, während der approximative Algorithmus für das selbe t und $\epsilon = 0.05$ nur 7.2 Sekunden brauchte.

Literatur

[1] Prosenjit Bose, Sergio Cabello, Otfried Cheong, Joachim Gudmundsson, Marc van Kreveld, and Bettina Speckmann. Area-preserving approximations of polygonal paths. *Journal of Discrete Algorithms 4*, pages 554–566, 2006.

[2] Paul Callahan and S. Rao Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. ACM 42*, pages 67–90, 1995.

[3] T.H. Cormen, C.E. Leiserson, R.L Rivest, and C. Stein. *Introduction to Algorithms.* 3rd edition, 2001.

[4] Joachim Gudmundsson, Giri Narasimhan, and Michiel Smid. Distance-preserving approximations of polygonal paths. *Computational Geometry 36*, pages 183–196, 2007.

[5] Torben Hagerup. *Vorlesungsskript Informatik III WS 17/18.* 2017.