Aggregation

Quellkodierung mit Zusatzinformation

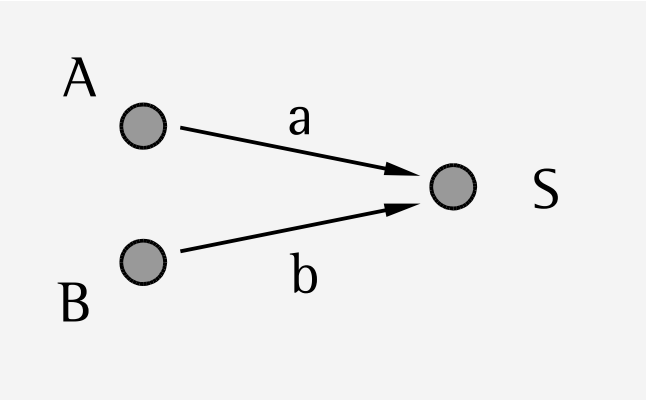
Aus dem Aufsatz „A Distributed and Adaptive Signal Processing Approach to Reducing Energy Consumption in Sensor Networks“ von J. Chou, D. Petrovic und K. Ramchandran.

Die Sensorknoten A und B senden Messwerte a und b an eine Senke S im Netzwerk. Aufgrund der räumlichen Nähe ist zu erwarten, dass die Messwerte Abhängigkeiten untereinander aufweisen, man spricht auch davon, dass sie korreliert sind.

Zuerst könnte A seinen Messwert an S sendet. Da der benachbarte Knoten B meist ähnliche Werte misst, würde es ausreichen statt das vollständigen b z. B. nur die Differenz a-b zu übertragen und damit Bits zu sparen. In der Senke S könnte man die Daten von B mit Hilfe der Messung von A trotzdem vollständig wiederherstellen.

Damit B differenziell kodieren kann könnte er bei A nach dessen Messwert fragen oder die Kommunikation zwischen A und S abhören, wenn Positionen und Sendereichweiten dies erlauben.

Auf den ersten Blick muss B scheinbar von a wissen, um Bits einsparen zu können. Man kann jedoch zeigen, dass B auch abhängig von A komprimieren kann, ohne jemands von A Daten zu empfangen.



Aggregation

Quellkodierung mit Zusatzinformation

Die spezielle Umsetzung für Sensornetze fußt auf der 1973 veröffentlichten Arbeit „Noiseless encoding of correlated information sources“ von D. Slepian und J. K. Wolf. Dabei entdeckten die Autoren, dass der Grad der Komprimierbarkeit einer Quelle in Abhängigkeit von einer anderen theoretisch keinen gegenseitigen Zugriff der Quellen aufeinander voraussetzt.

$$H(B|A) = \sum_{\forall a} \left[P_A(a) \sum_{\forall b} P_B(b|a)(-1)\log_2\left(P_B(b|a)\right) \right]$$

Anzahl der Bits die benötigt werden, um den Zustand „b unter Voraussetzung von a“ zu kodieren. Große Wahrscheinlichkeit -> keines log, keine Wahrscheinlichkeit, großes log. Warum? Weil häufige Fälle mit wenigen Bits kodiert werden müssen.

Wahrscheinlichkeit dass sich b realisiert, wenn gerade a vorliegt. Dies kann durchaus öfter Null sein.

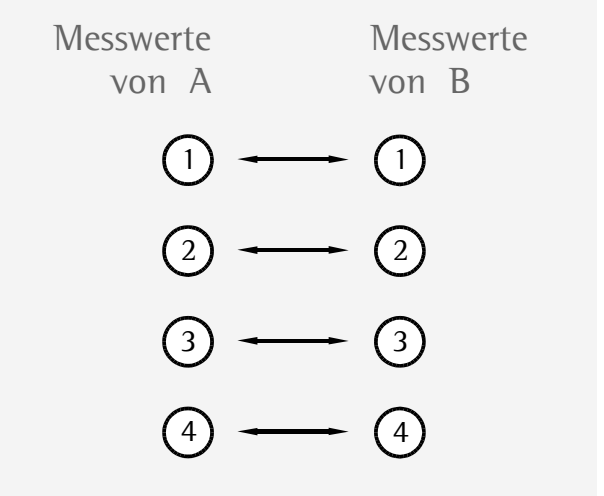
Summe über alle Möglichkeiten, die sich für Variable A realisieren können. Jedes mögliche a tritt mit einer Wahrscheinlichkeit P(a) auf.

Anzahl der Bits die benötigt werden um Zufallsvariable B in Abhängigkeit von A zu kodieren. Lies „B unter der Voraussetzung A“.

Aggregation

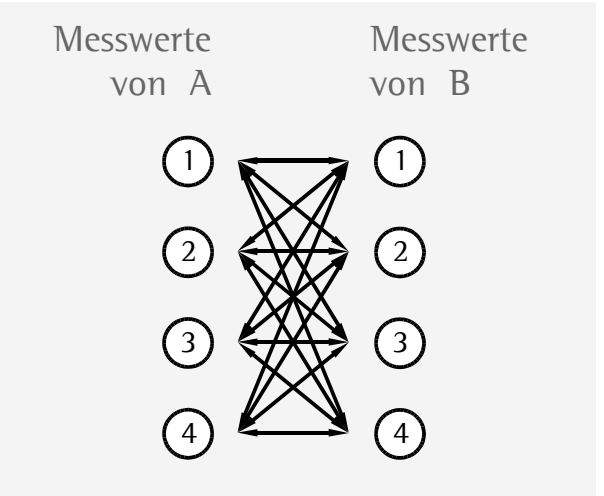
Quellkodierung mit Zusatzinformation

Beispiel 1: Die Quelle B misst immer exakt den gleichen Sachverhalt wie die Quelle A. Hier ist nicht verwunderlich, dass die Senke S unter Kenntnis von A die Messung von B ermitteln kann. B braucht hierzu keine Kenntnis des Wertes von A zu haben. In diesem Fall muss B der Senke auch überhaupt keine Bits zuschicken.



Beispiel 2: Quelle A und B würfeln ihre Werte aus. Beide Zufallsvariablen sind also vollkommen unkorreliert. Für jeden der vier möglichen Werte von A ist jeder der vier Werte des Wertebereichs von B gleich wahrscheinlich.

Die Kenntnis des Wertes von A hilft B offensichtlich nicht weiter und kann daher ebenso bekannt, wie unbekannt sein.



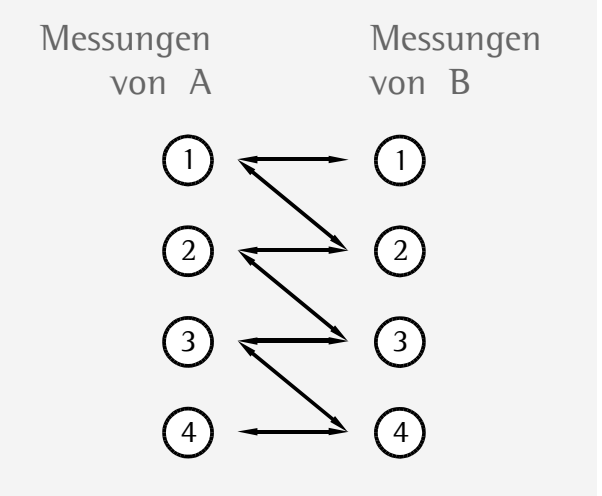
Aggregation

Quellkodierung mit Zusatzinformation

Beispiel 3: Die Quelle B misst manchmal den gleichen Wert wie A , manchmal einen um 1 erhöhten Wert.

A überträgt seinen Messwert vollständig an die Senke, B teilt lediglich mit ob sein Messwert gerade oder ungerade ist, braucht also 1 Bit. Auch hier greift immer noch der Sachverhalt, dass B und A nichts voneinander wissen.

In der Senke kommt von A der Wert 2 an. Dies bedeutet, dass B entweder eine 2 oder eine 3 gemessen haben muss. Daher muss B nur noch die Parität angeben, damit seine Messung dekodierbar ist.



Aggregation

Quellkodierung mit Zusatzinformation

$$H(B|A)=\sum_{\forall a}\left[P_A(a)\sum_{\forall b}P_B(b|a)(-1)\log_2\left(P_B(b|a)\right)\right]$$

A=	1		2		3		4
P(A)=	0.25		0.25		0.25		0.25
B|A=	1	2	2	3	3	4	4
P(B|A)=	0.5	0.5	0.5	0.5	0.5	0.5	1
-log(P(B|A))=	1	1	1	1	1	1	0
-P(B|A)log(P(B|A))=	0.5	0.5	0.5	0.5	0.5	0.5	0
Sum [P()P()log()]=	1		1		1		0
H(B|A)=	0.75 (Bit) für die kodierung von B						

Theoretisch wäre für die Kodierung von B in Abhängigkeit von A also nur 0.75 Bit nötig. Die Konvention auf der vorherigen Seite besagt, dass B immer die Parität seines Messwertes verschickt. Somit werden tatsächlich 1 Bit pro Wert von B verbraucht.

Man könnte die Konvention dahingehend erweitern, dass B keine Information verschickt, wenn es den Wert 4 übertragen möcht. Da ein Wert 5 nicht existiert, kann die Senke aus A=4 schließen, dass auch B=4 gelten muss. Somit wären die 0.75 Bit nach der Abschätzung von Slepian und Wolf auch praktisch erreicht.

Es ist nicht immer offensichtlich, wie ein Code definiert werden kann, der das theoretische Minimum auch tatsächlich erreicht. Der Huffman-Code ist z. B. immer nur dann optimal, wenn die relativen Häufigkeiten der Zeichen eines Code aus der Menge 2^{-n} stammen (und n eine natürliche Zahl ist).

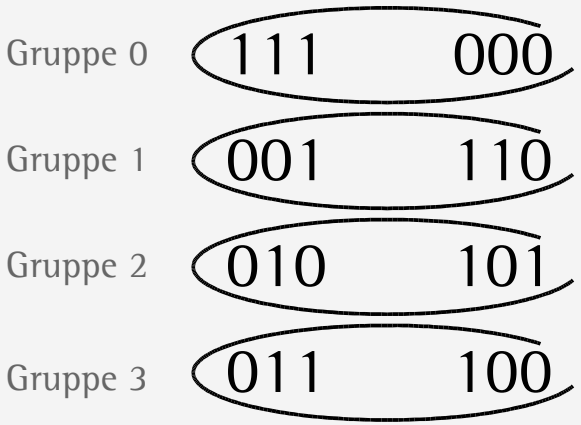
Aggregation

Quellkodierung mit Zusatzinformation

Beispiel 4: Für zwei 3-Bit Zufallsvariablen d_A und d_B ist bekannt, dass ihr Hamming-Abstand <= 1 ist.

Nachdem A seinen 3-Bit Wert unkodiert an die Senke gesendet hat steht fest, dass sich d_B nur um ein einzelnes Bit unterscheiden darf bzw. dass gilt $d_B=d_A$.

Um diese Information mitzuteilten, muss Knoten B vier Zustände kodieren können, d. h. zwei Bits verbrauchen. Ein drittest Bit ist offensichtlich nicht nötig, so dass ein Bit gespart wurde. Bei größeren Wertebereichen als 3 Bit wäre die Ersparnis entsprechend größer. Es geht nun darum vier Gruppen zu bilden, so dass Gruppe B nur noch die Gruppe bezeichnen muss. Durch den Wert von A ist dann das Element der bezeichneten Gruppe bereits eindeutig bestimmt.



Beispiel: Knoten A hat den Wert 110 gemessen, Knoten B den Wert 100. Da 100 in Gruppe 3 enthalten ist, sendet B eine binäre 3, also 11.

Die Senke erhält die Information, dass der Wert von B in Gruppe 4 enthalten sein muss. Es kommt also 100 oder 011 für B in betracht. 011 hat jedoch von 110 den Hammingabstand 2, 100 hat von 110 nur den Abstand 1. Somit weiß die Senke, dass B den Wert 100 gemessen hat.

Aggregation

Quellkodierung mit Zusatzinformation

Warum muss B die Messungen von A nicht kennen?

B „benennt“ oder formaler „kodiert“ seine eigenen Werte in Abhängigkeit von A. Da für jeden Wert aus B in Beispiel 3 nur zwei von A in Frage kommen, reicht 1 Bit aus. Je weniger korreliert die beiden Datenquellen sind, desto mehr Werte von A kommen für jedes B in Frage. Im äußersten Fall kommen für alle Werte von B alle von A in Frage, so dass B seine Messung vollständig selbst übertragen muss.

Die Kodierung mit Zusatzinformation (in der Literatur „Coding with side information“) wirkt weniger verblüffend sobald man sich klar macht, dass B in Wirklichkeit (je nach Stärke der Korrelation mit A) nur wenig „echte“ Information enthält. Die meiste Information hat B möglicherweise mit A gemeinsam. Wenn dieser Sachverhalt global bekannt ist, muss B die Werte von A nicht kennen. Anschaulich gesprochen muss B also wissen welcher Anteil seiner Information „unsicher“ und welcher deterministisch (= die Zusatzinformation) ist. Wenn dies bekannt und über die Zeit stabil ist, kann sich B auf die Übertragung der nicht-redundanten, unsicheren Information beschränken.

Problem in Sensornetzen: B soll in Abhängigkeit aller anderen Knoten kodieren, kennt deren Messungen aber nicht. Darüber hinaus muss die Frage beantwortet werden, wie Messwerte in Gruppen zusammengefaßt werden.

Aggregation

Quellkodierung mit Zusatzinformation

Grundsätzliche Vorgehensweise, unabhängig von der Korrelation zweier Quellen:

Bilde Gruppen aus Wörtern bzw. Nachrichten, die paarweise einen möglichst großen Abstand haben müssen. Quelle B gibt dann nur die Gruppe an, aus der ihr Wert stammt. Zusammen mit dem vollen Wert von A kann dasjenige Element der Gruppe gefunden werden, welches den kleinsten Abstand zu A hat. Dieses Element ist dann der Wert von B.

Das Verfahren funktioniert grundsätzlich immer dann, wenn Quelle B von A nur um ein begrenztes Maß abweicht.

Veranschaulichung: A und B seien Punkte in einer Ebene. Dabei ist B mit A über ein Band verbunden, dass nur eine begrenzte Entfernung zuläßt. B muss sich also immer in einem Umkreis von A befinden. Dieser Anteil der Information ist global bekannt, stellt keine Unsicherkeit dar und muss nicht kodiert werden. Lediglich der genaue Ort im Umkreis von A ist unsicher. Nur dieser muss übertragen werden.

Der „Trick“ besteht nun darin, alle Orte die A annehmen könnte in Gruppen zusammen zu fassen, die so weit wie möglich voneinander entfernt sind. B wählt dann die Gruppe aus, in der seine Position (exakt) enthalten ist. Der Senke wird nur eine Referenz auf die Gruppe mitgeteilt. Im Gegensatz zu B hat die Senke den evtl. erheblichen Aufwand alle Elemente der Gruppe mit der Position von A zu vergleichen. Das Element mit dem kleinsten Abstand zu A muss die Position von B sein. Warum? Weil alle anderen Elemente der Gruppe so weit von B entfernt sind, dass die max. Distanz die B von A haben darf verletzt wird. Genau so wurden die Gruppen nämlich zusammengestellt.

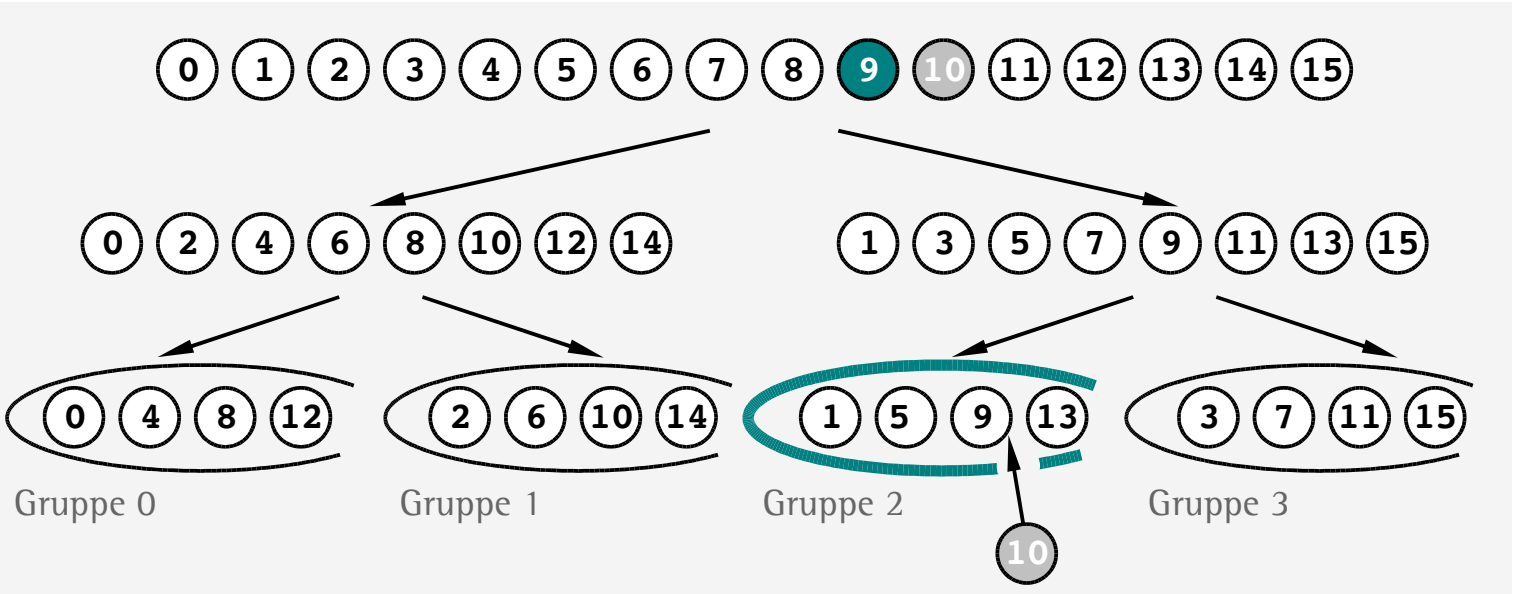
Aggregation

Quellkodierung mit Zusatzinformation

Anwendung für Sensorknoten:

Ein n-Bit Analog-Digitalwandler erzeugt 2^n Werte. Im Beispiel unten darf B sich um max. +/- 1 von A unterscheiden. B mißt den Wert 9 und wählt die Gruppe 2 aus. Die Senke erhält diese Information von B und die vollständige Messung 10 von A.

Da 9 zu 10 den kleinsten Abstand hat, kann die Senke daraus B=9 schließen.



Aggregation

Vektorquantisierung

Bei einem ganzen Messvektor wäre die sog. Vektorquantisierung sinnvoll, insbesondere dann, wenn ein Knoten evtl. die Daten mehrerer Kinder sammeln muss, bevor er sie weiterleitet. Sehr gut auch mit dem TAG-Ansatz kombinierbar.

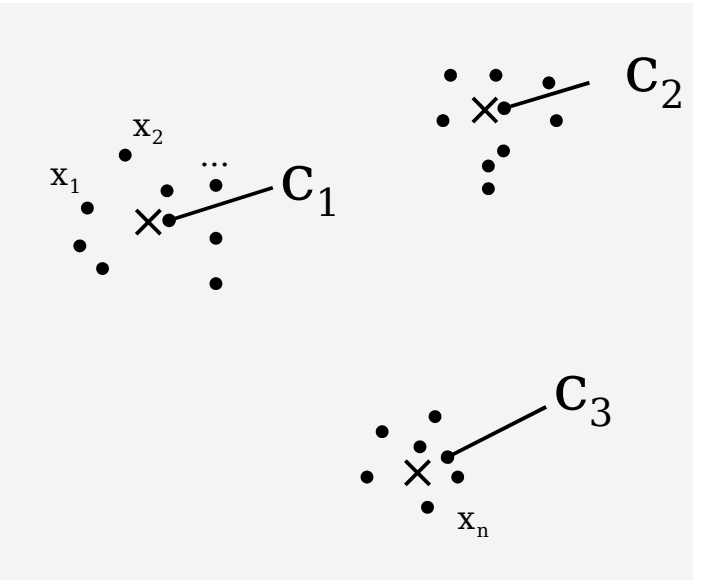
Idee: Es existiert eine (große) Testmenge X von Vektoren

$$X = \{x_1, x_2, \ldots, x_n\}$$

gesucht ist nun eine Menge C von Vektoren

$$C = \{c_1, c_2, \ldots, c_m\}$$

... das sog. Codebook. Die Kompression besteht später nur noch darin, den Index eines Repräsentanten des Codebooks zu speichern (bzw. zu übertragen). Bei einer gegebenen Anzahl von Einträgen des Wörterbuchs stellt sich die Frage, wie diese so im Raum der Sampelwerte verteilt werden müssen, damit ein möglichst kleiner mittlerer Fehler durch die Approximation entsteht.



Aggregation

Vektorquantisierung

für die Auswahl eines Eintrages des Codebooks muss folgende Bedingung gelten:

$$I_x = arg(min_{\forall i} |x - c_i|)$$

d. h. es wird über alle Einträge des Codebook iteriert. Derjenige Eintrag mit der kleinsten Differenz zu einem bel. Wert (nicht zwingend aus dem Testset, d. h. die Einträge des Codebooks müssen nicht unbedingt Werte sein, die bereits früher beobachtet wurden) „gewinnt“ die Minimierung. Dabei ist nicht in erster Linie die eigentliche Differenz von Interesse. Statt dessen zieht der arg-Operation das Argument (hier Index i) heraus, für das das Minimum angenommen wurde.

Insgesamt soll das Codebook theoretisch nach dem folgendn Kriterium optimiert werden.

$$(c_1, \ldots, c_m) = arg(min_{\forall c_1, \ldots, c_m} \sum_{i=0}^{n} |c_{I_x} - x_i|)$$

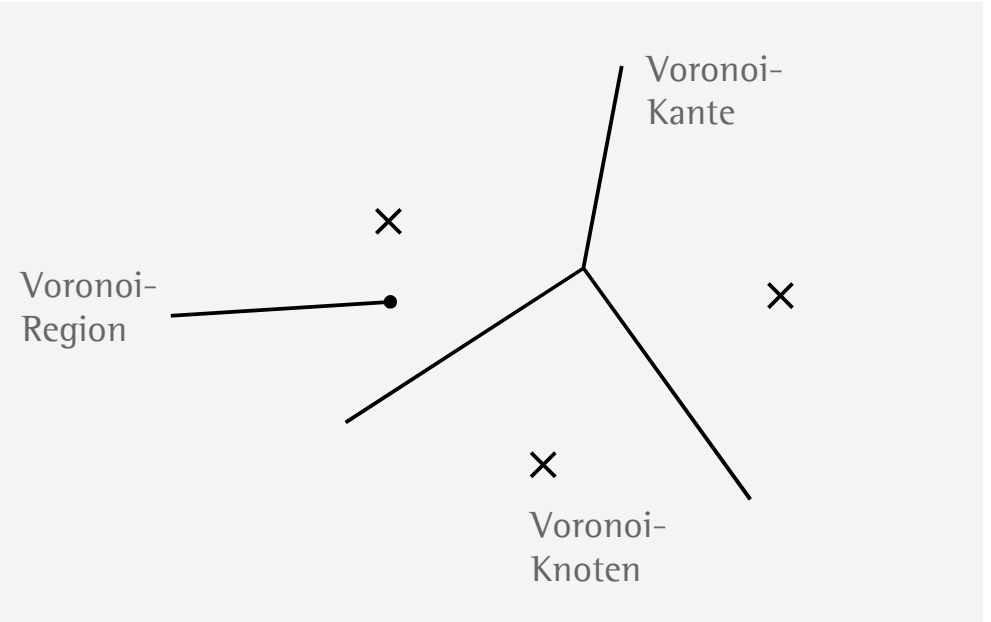
Leider ist die Ermittlung optimaler Codebook-Einträge ein NP-hartes Problem. Es existieren jedoch Algorithmen die meist lokale Optima finden.

Aggregation

Vektorquantisierung: Voronoi Diagramme

Das Codebook der Vektorquantisierung entspricht hier den sog. Voronio-Knoten. Diese prepräsentieren den gesamten Vekor- oder Unterraum, der durch die Sensoren erfaßt werden kann. Im Folgenden werden o.B.d.A. Beispiele in der Ebene gezeigt.

Die Voronoi-Kanten teilen den Raum in teilweise offene, teilweise geschlossene konvexe Polyeder, die Voronoi-Regionen auf. Jeder Punkt innerhalb eines Polyeders wird durch dessen Knoten mit dem kleinsten Fehler approximiert.

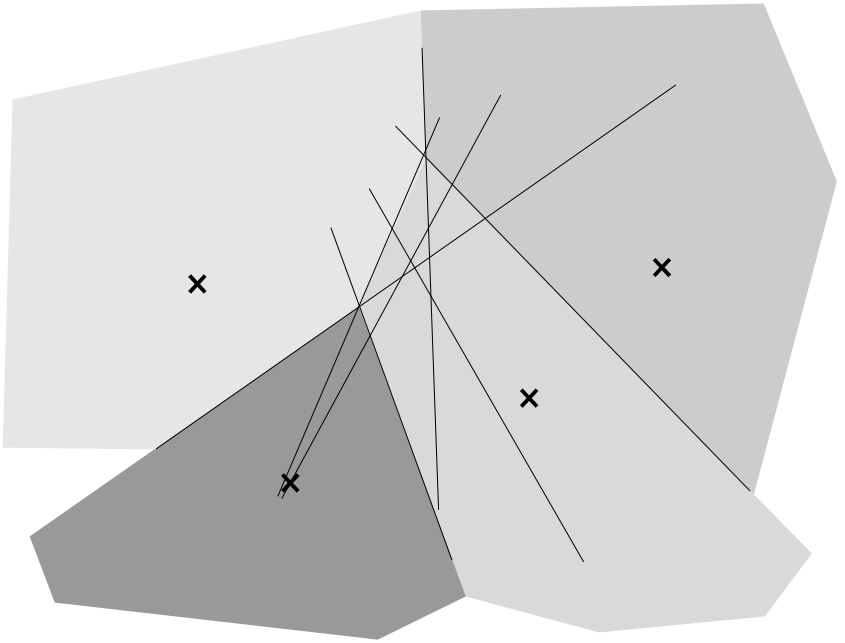


Aggregation

Vektorquantisierung: Generierung von Voronoi Diagrammen (naiv)

Das aus einer Menge von Punkten bestehende Codebook sei bereits gegeben. In einer naiven Implementierung könnte man zwischen allen möglichen Paaren von Punkten Mittelsenkrechten errichten, die den Raum in jeweils zwei Halbräume teilen. Durch die vielen Schnitte der Mittelsenkrechten entstehen Polyeder, die noch jeweils ihrem räumlich nächsten Punkt zugeordnet werden müssen. Bereiche gleicher Zugehörigkeit können am Ende vereinigt werden.

Nachteil: Großer Aufwand, durch n^2 Schnitte.



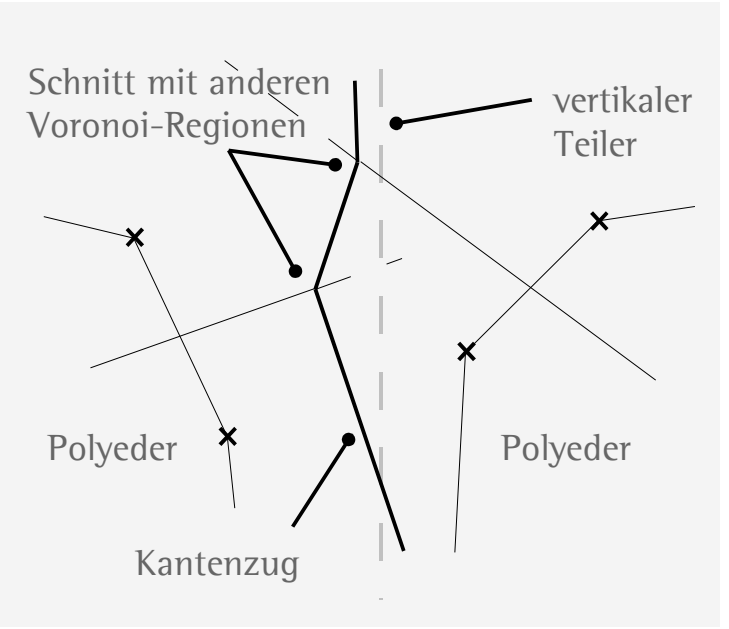
Aggregation

Vektorquantisierung: Generierung von Voronoi Diagrammen (devide and conquer)

(1) **Teile die Knotenmenge P durch einen vertikalen (oder horizontalen) Teiler** in zwei Hälften P_a und P_b. Dabei sollte gelten $||P_a|-|P_b||$ -> min (möglichst gleichgroße Mengen).

(2) **Teile P_a und P_b rekursiv in kleinere Mengen.** Die Rekursion endet, wenn nur noch ein Punkt in der Menge verbleibt. Die dem Punkt zugeordnete Region ist die gesamte Ebene.

(3) **Vereinigen der zwei Teildiagramme** von P_a und P_b: Beide Punktmengen haben zwangläufig eine konvexe Hülle. Nähert man sich diesen von oben kommend, so erreicht man einen obersten Punkt. Dann fällt man eine Mittelsenkrechte auf die Verbindung dieser beiden Punkte. Diese ist bereits die anfängliche obere Halbgerade des Kantenzuges, der P_a und P_b trennt. Es folgen einige Geradenstücke und am Ende schließt der Kantenzug wieder mit einer Halbgeraden ab.



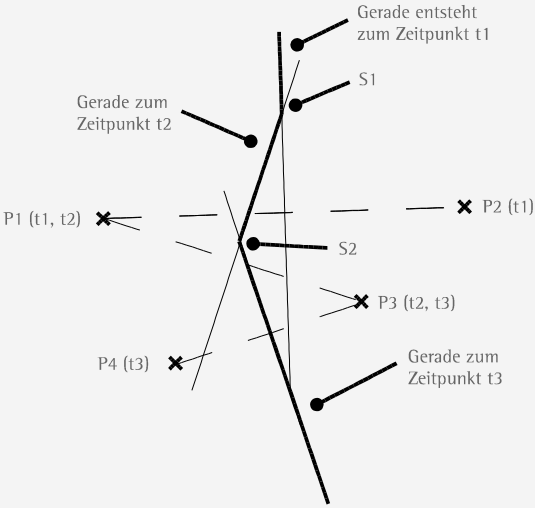
Aggregation

Vektorquantisierung: Voronoi Diagramme (Ermitteln des Kantenzuges K)

Zeitp. t1: Die obersten Knoten P_1 und P_2 der jeweiligen konvexen Hülle P_a und P_b sind aktiv. Auf die Verbindungsgerade wird ein Lot gefällt.

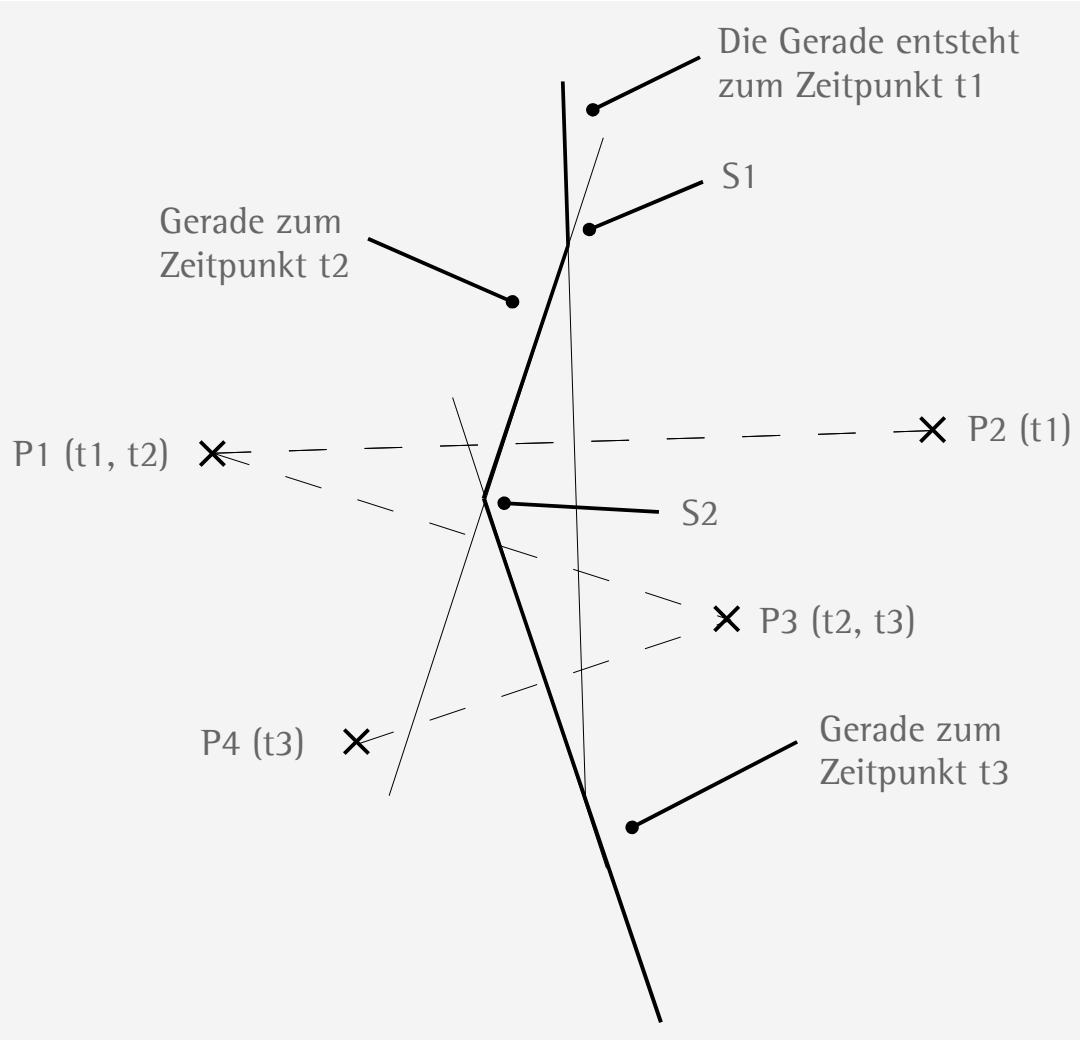
Zeitp. t2: Unterhalb von Schnittpunkt S_1 ist das Lot näher an P_3 als an P_1. Daher wird P_2 zugunsten von P_3 abgelöst. Wieder setzt die Mittelsenkrechte zwischen P_1 und P_3 den Kantenzug K fort.

Zeitp. t3: Unterhalb von Schnittpunkt S_2 ist das Lot näher an P_4 als an P_1, daher wird P_1 zu gunsten von P_4 abgelöst.



Aggregation

Vektorquantisierung: Voronoi Diagramme (Ermitteln des Kantenzuges K)



Aggregation

Directed diffusion

Aus der Veröffentlichung „Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks“ von C. Intanagonwiwat, R. Govindan, D. Estrin.

Knoten werden nicht als Individuen addressiert, sondern durch ihre Fähigkeit, ihren Ort etc. Bekannt sind jedoch Eltern-Kind Beziehungen in beide Richtungen. Jedoch gibt es keine IDs. Auch Beziehungen zweiten Grades sind bereits nicht mehr bekannt.

Annahmen:

Es gibt eine Datensenke, jeder Knoten ist eine potenzielle Datenquelle

Ein Knoten weiß, wo er sich geographisch befindet. Er kennt auch die Region, in denen sich seine Kinder befinden. Diese wird durch ein achsenparalleles Rechteck eingegrenzt.

Es kann eine zufällige Knotentopologie entstehen die von einer Gleichverteilung in einem Kreis bis zu einer linearen Ausbringung reichen kann.

Ein Knoten kann skalare Sensordaten sampeln und evtl. gleich mit gespeicherten Mustern vergleichen. Möglicherweise wird nur das Ergebnis des Vergleichs an den Vater-Knoten weitergegeben.

Ein Knoten kennt seine unmittelbaren Nachbarn. Routing erfolgt in kurzen Distanzen hop-by-hop. Dadurch können (1) Daten auf dem Weg zur Senke bereits aggregiert werden und (2) evtl. Hinternisse um-routet werden.

Aggregation

Directed diffusion

Beispiel: Detektiere ein vierbeiniges Tier in der Region (x1,y1,x2,y2) und schicke alle 1000ms ein Ergebnis in Richtung der Senke.

Sensoren könnten bzw. müssen evtl. mehrere Aufgaben erfüllen. Aufgaben (oder die sog. „Interests“) bestehen im Wesentlichen aus einfachen (Attribut, Wert) Paaren. Der `timestamp` gibt den Zeitpunkt der Erzeugung der Anfrage an, die `expiration` Zeit das Ende der Anfrage. Von den Sensoren die das Ereignis detektieren können wird alle 20ms eine Messung erwartet:

Anfrage bzw. Interest:

```
type        = four-legged animal
interval    = 200ms
timestamp   = 01:20:40
expiration  = 01:30:40
rectangle   = [-100, 100,
               200, 400]
```

Antwort:

```
type        = four-legged animal
instance    = elephant
location    = [70, 220]
intensity   = 0.6
confidence  = 0.85
timestamp   = 01:25:10
```

Aggregation

Directed diffusion

In „Directed diffusion“ sollen Sensoren anhand einer Eigenschaft adressiert werden, also z. B. anhand der Eigenschaft ein vierbeiniges Tier zu detektieren etc. Welche Knoten dies sein werden, ist anfangs nicht klar.

Initialisierung: Die Anfrage wird mit einer großen Intervallzeit (gefordert werden nur wenige Samples/Sekunde) im Netz verteilt. Dabei soll sich herausstellen, welche Knoten die Anfrage überhaupt positiv beantworten können. Dazu sendet die Datensenke die Anfrage an ihre Nachbarn. Ist der abgefragte Bereich geographisch eingegrenzt, so können alle Nachbarn gefiltert werden, die nicht selbst bzw. deren Kinder nicht im angefragten Bereich liegen. Ansonsten entspricht die Initialisierung einem Fluten des Netzes.

Ein Knoten der eine Anfrage erhält, trägt diese in eine Tabelle ein, falls sie nicht bereits enthalten ist.

```
type       = four-legged animal
rectangle  = [-100, 100, 200, 400]
gradient[]  // je ein Eintrag pro Vater
  .datarate // entnommen aus interval der Anfrage des Vaters
  .duration // expiration-timestamp der Anfrage des Vaters
```

Jede Anfrage kennt eine Menge von „Gradienten“. Damit meinen die Autoren Referenzen auf den Nachbarknoten, von dem die Anfrage gestellt wurde. Der anfragende Knoten muss auch spezifizieren, mit welcher Datenrate (`.datarate`) und bis zu welchem Zeitpunkt (`.duration`) er Ereignisse erhalten will.

Aggregation

Directed diffusion

Initialisierung Fortsetzung: Kommt eine bereits registrierte Anfrage von einem weiteren Knoten, so wird nur ein neuer Gradient für den entsprechenden Interest eingetragen. Der Knoten selbst benötigt also die Daten (spezifiziert durch den Interest) in der höchten Rate aller seiner Gradienten. Wenn er die Daten selbst nicht erzeugen kann, muss er sie seinerseits von anderen Knoten anfordern. Mehr Gradienten als Nachbarn kann es nicht geben.

Die Tabelle der Interests verhindert vor allem dass Zyklen entstehen, in denen Anfragen endlos zirkulieren. Außerdem kann der gleiche Datenstrom in unterschiedlichen Intervallen „nach oben“, also in Richtung der Datensenke(n) weitergegeben werden.

Initialer Datenfluss: Ein Knoten sampelt Daten. Wenn diese zur Charakteristik eines Interests passen, werden sie an alle registrierten Teilnehmer über die Gradienten-Einträge weitergegeben. Ein Knoten der selbst keine Daten mißt, sondern Messungen anderer Knoten weiter nach oben gibt, braucht einen Daten-Cache. Sonst würden sich Daten ebenso wie Interests im Netz grenzenlos vermehren. Der initiale Datenfluss wird möglicherweise viele Duplikate und Messungen des gleichen Ereignisses durch evtl. mehrere Knoten bei der Senke erzeugen. Der initialen Phase schließt sich dann eine...

Reinforcement-Phase (Verstärkung) an. Die anfängliche Anfrage wird nach Ablauf der expiration Zeit verschwinden. Die Senke ist nun daran interessiert eine hohe Datenrate von dem Knoten zu erhalten, der das Ereignis ab besten (oder überhaupt) detektieren kann, und dies auf dem kürzesten Weg. Die Senke kann ebenfalls annehmen, dass das Paket, welches zuerst angekommen ist auch auf dem optimalen (weil kürzesten oder schnellsten) Pfad zum Ereignis liegt. Sie wird daher eine explizite Anfragen an diesen Nachbarknoten richten.

Aggregation

Directed diffusion

Reinforcement-Phase Fortsetzung: Der von der Senke angesprochene Knoten stellt seinerseits eine ähnliche Anfrage an den Knoten, der seinen Interest zuerst mit Daten bedienen konnte usw. Dadurch entsteht nach einer aufwändigen Flooding-Phase der kürzeste Weg zur Ereignis bzw. dem Knoten, der es auf dem optimalen Weg detektieren konnte.

Kritik:

Wie wird verhindet, dass jeder Knoten jeden anderen als Gradienten registriert?

Ist es realistisch, dass in der Phase des initialen Datenflusses ein Knoten alle Daten zwischenspeichern kann, die er bereits gesehen hat? Dies ist unabdingbar um Duplikate zu vermeiden, ist aber schlecht mit den begrenzten Speicherressourcen von Sensorknoten zu vereinbaren. Man müßte den Cache von Zeit zu Zeit leeren oder als Ringspeicher organisieren. Es ist jedoch nicht klar, ab welchem Zeitpunkt ein Datum verworfen werden kann.

Worin liegt die Motivation, dass eine Messgröße innerhalb des Netzes mit unterschiedlichen Datenraten angefordert wird?

Das gesamte Netz muss/sollte sich einer dedizierten Aufgabe widmen, da durch das Flooding ohnehin alle Knoten zumindest bei der Initialisierung mit einer Anfrage beschäftigt werden.

Nach den Autoren erfordert ein bewegliches Ereignis die fortlaufende Initialisierung des Netzes.