Multiagentensysteme

Technologien, Architekturen, Plattformen

Prof. Dr. Stefan Bosse

Universität Koblenz - FB Informatik - Praktische Informatik

Verteiltes Verhalten und Gruppen

Gruppenentscheidung und Verhandlung

- In Multiagentensystemen gibt es in der Regel **Organisationsstrukturen**
- In diesen Strukturen sollen **gemeinsame Ziele** entweder vorgegeben und umgesetzt oder verhandelt werden.

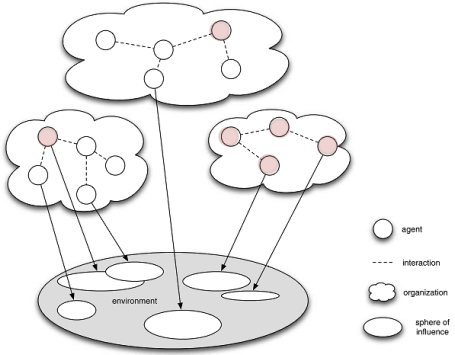


Abb. 1. Typische Gruppenstrukturen in Multiagentensystemen

Gruppenentscheidung und Verhandlung

- Dabei können Gruppenentscheidungen und Verhandlungen auf Basis von **Nützlichkeitsfunktionen** geführt werden
- Es sei eine Gruppe von Agenten (Prozessen) $G=\{ag_1,ag_2,..,ag_m\}$
- Es gibt nun verschiedene Stufen der Verhandlung:
  - Wahl eines Gruppenführers (Leader) oder Mediators
  - Abgabe von Stimmen / Absichten
  - Bestimmung eines gemeinsamen Ergebnisses mittels Konsensalgorithmus bzw.
  - Wahl: Über Pluralität, z.B. mit einer Mehrheitsentscheidung
  - Benachrichtigung der Gruppenteilnehmer über Ergebnis
- **Nachteil des Mehrheitsentscheids**: Das Volk ist dumm! D.h. es könnte eine besseres Ergebnis für ein MAS erzielt werden, wenn Fraktionen in den Stimmabgaben berücksichtigt werden würden (differenziertes und gewichtetes Ergebnis) ...

Verhandlung und Abstimmung

- Verhandlung in Gruppen und Abstimmung über ein gemeinsames Ergebnis (Konsens) ist ein weiteres wichtiges Beispiel für verteiltes Gruppenverhalten → Kommunikation!
- Ein Verhandlungsproblem ist ein Problem, bei dem mehrere Agenten versuchen, zu einer Vereinbarung oder einem Deal zu kommen.
- Es wird angenommen, dass jeder Agent gegenüber allen möglichen Deals eine Präferenz hat.
- Die Agenten senden sich Nachrichten in der Hoffnung, einen Deal zu finden, auf den sich alle Agenten einigen können.
- Diese Agenten stehen vor dem Problem:
  - Sie wollen ihren eigenen Nutzen maximieren, sehen sich aber auch der Gefahr eines Zusammenbruchs der Verhandlungen oder des Ablaufs einer Frist für die Vereinbarung gegenüber.
  - Daher muss jeder Agent sorgfältig verhandeln und jeden Nutzen abwägen, den er aus einem Versuch gegen einen möglicherweise besseren Abschluss oder das Risiko eines Ausfalls bei den Verhandlungen zieht.

Verhandlung und Abstimmung

> Automatisierte Aushandlung kann in Multiagentensystemen sehr nützlich sein, da sie eine verteilte Methode zur Aggregation von verteiltem Wissen bietet.

- Verschiedene Protokolle existieren, z.B.,
  - Monotone Konzession
  - Monotone Konzession mit zusätzlicher Risikobewertung
  - Schrittweise Verhandlung

- Häufig in der Form (Monotone Konzession, Vidal,2010)

```
0. δi <- arg maxδ ui(δ)
1. Propse δi
2. Receive δj proposal
3. if ui(δj) > ui(δi)
4.   then Accept δj
5.   else δi <- δi' such that uj(δi') >= e+uj(δi)
6. loop 2.
```

- mit $u_i(\delta)$: Nützlichkeitsfunktion eines Deals δ des i-ten Agenten

Verteilter Konsens

- Ein verteilter Konsensalgorithmus hat das Ziel in einer Gruppe von Prozessen oder Agenten eine gemeinsame Entscheidung zu treffen
- Zentrale Eigenschaften:
  - Zustimmung/Übereinstimmung
  - Terminierung; Lebendigkeit und Deadlockfreiheit
  - Gültigkeit; Robustheit gegenüber Störungen wie fehlerhaften Nachrichten oder Ausfälle von Gruppenteilnehmern
- Beim Konsens kann ein Master-Slave Konzept oder ein Gruppenkonzept mit Leader/Commander und Workern verwendet werden.
  - Beim Master-Slave Konzept kommunizieren nur Slaves mit dem Master
  - Bei Gruppenkonzept (i.A. mit einem Leader) kommunizieren auch alle Gruppenteilnehmer untereinander
- Durch Störung (Fehler oder Absicht) kann es zu fehlerhaften bis hin zu fehlgeschlagenen Konsens kommen.

Verteilter Konsens

- Bedingungen für Interaktive Konsistenz:
  - IC1: Jeder Worker empfängt die *gleiche* Anweisung vom Leader!
  - IC2: Wenn der Leader *fehlerfrei* arbeitet, dann empfängt jeder *fehlerfreie* Worker die Anweisung die der Leader sendete!

Byzantinisches Generalproblem

- Beispiel: In einer Gruppe aus drei Prozessen/Agenten ist einer fehlerhaft bzw. versendet fehlerhafte Nachrichten (durch Störung oder Absicht) mit Anweisungen 0/1 (schließlich ein Konsensergebnis) [E]

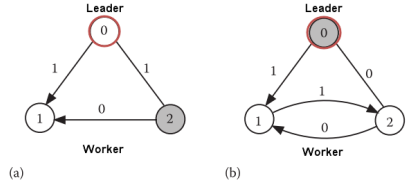


Abb. 2. Byzantinisches Generalproblem: (a) Leader 0 ist fehlerfrei, Worker 2 ist fehlerhaft (b) Leader 0 ist fehlerhaft, Worker 1 und 2 sind fehlerfrei [E]

Verteilter Konsens

- Jeder Worker der Nachrichten empfängt ordnet diese nach direkten und indirekten (von Nachbarn)
- **Fall (a)**: Prozess 2 versendet fehlerhafte Nachricht mit Anweisung 0, Prozess 1 empfängt eine direkte Nachricht mit Anweisung 1 und eine indirekte mit (falschen) Inhalt Anweisung 0
  - Bedingung IC1 ist erfüllt. Um Bedingung IC2 zu erfüllen wird Worker 1 die direkte Anweisung 1 von Prozess 0 (Leader) auswählen → Konsens wurde gefunden
- **Fall (b)**: Prozess 0 (Leader) versendet an Prozess 1 richtige Nachricht mit Anweisung 1 und falsche Nachricht mit Anweisung 0 an Prozess 1
  - Würde Prozess 1 wieder zur Erfüllung von IC2 eine Entscheidung treffen (Anweisung 1 auswählen), dann wäre IC1 verletzt. Wie auch immer Prozess 1 entscheidet ist entweder IC1 oder IC2 verletzt → **Unentscheidbarkeit** → Kein Konsens möglich

Verteilter Konsens

Das nicht-signierte Nachrichtenmodell erfüllt die Bedingungen:

1. Nachrichten werden während der Übertragung nicht verändert (aber keine harte Bedingung).
2. Nachrichten können verloren gehen, aber die Abwesenheit von Nachrichten kann erkannt werden.
3. Wenn eine Nachricht empfangen wird (oder ihre Abwesenheit erkannt wird), kennt der Empfänger die Identität des Absenders (oder des vermeintlichen Absenders bei Verlust).

- Algorithmen zur Lösung des Konsensproblems müssen *m* fehlerhafte Prozesse annehmen (bzw. fehlerhafte Nachrichten)

Der OM(m) Algorithmus

- Ein Algorithmus der einen Konsens erreicht bei Erfüllung der Bedingungen IC1 und IC2 mit bis zu *m* fehlerhaften Prozesse bei insgesamt $n \geq 3m+1$ Prozessen mit nicht signierten ("mündlichen") Nachrichten.
  i. Leader *i* sendet einen Wert $v \in \{0, 1\}$ an jeden Worker $j \neq$ i. ii. Jeder Worker *j* akzeptiert den Wert von *i* als Befehl vom Leader *i*.

Verteilter Konsens

Def. 1. Algorithmus OM(m)

1. Leader i sendet einen Wert $v \in \{0, 1\}$ an jeden Worker $j \neq$ i.
2. Wenn $m > 0$, dann beginnt jeder Worker j, der einen Wert vom Leader erhält, eine neue Phase, indem er ihn mit OM(m-1) an die verbleibenden Worker sendet.
   - In dieser Phase fungiert j als Leader.
   - Jeder Arbeiter erhält somit (n-1) Werte: (a) einen Wert, der direkt von dem Leader i von OM (m) empfangen wird und (b) (n-2) Werte, die indirekt von den (n-2) Workern erhalten werden, die aus ihrem Broadcast OM(m-1) resultieren.
   - Wird ein Wert nicht empfangen wird er durch einen Standardwert ersetzt.
3. Jeder Worker wählt die Mehrheit der (n-1) Werte, die er erhält, als Anweisung vom Leader i.

Verteilter Konsens

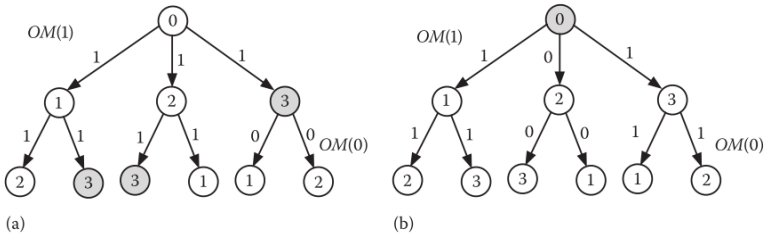


Abb. 3. Eine Illustration von OM(1) mit vier Prozessen und einem fehlerhaften Prozess: die Nachrichten auf der oberen Ebene spiegeln die Eröffnungsnachrichten von OM(1) wider und die auf der unteren Ebene spiegeln die OM(0)-Meldungen wider, die von den Mitteilungen der oberen Ebene ausgelöst werden. (a) Prozess 3 ist fehlerhaft. (b) Prozess 0 (Leader) ist fehlerhaft. [E]

Verteilter Konsens

Der Paxos Algorithmus

- Paxos ist ein Algorithmus zur Implementierung von fehlertoleranten Konsensfindungen.
- Er läuft auf einem *vollständig verbundenen Netzwerk* von n Prozessen und toleriert bis zu m Ausfälle, wobei $n \geq 2m+1$ ist.
- Prozesse können abstürzen und Nachrichten können verloren gehen, byzantinische Ausfälle (absichtliche Verfälschung) sind jedoch zumindest in der aktuellen Version ausgeschlossen.
- Der Algorithmus löst das Konsensproblem bei Vorhandensein dieser Fehler auf einem *asynchronen System von Prozessen*.
- Obwohl die Konsensbedingungen Zustimmung, Gültigkeit und Terminierung sind, garantiert Paxos in erster Linie die Übereinstimmung und Gültigkeit und nicht die Beendigung - es ermöglicht die Möglichkeit der Beendigung nur dann, wenn es ein ausreichend langes Intervall gibt, in dem kein Prozess das Protokoll neu startet.

Verteilter Konsens

- Ein Prozess kann drei verschiedene Rollen wahrnehmen:
  - Antragsteller,
  - Akzeptor und
  - Lerner.

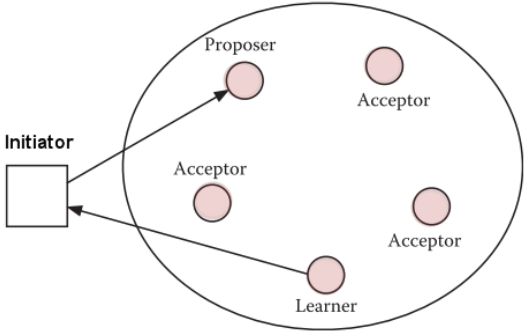


Abb. 4. Typische Rollenverteilung beim Paxos Algorithmus

Verteilter Konsens

- Die **Antragsteller** reichen die vorgeschlagenen Werte im Namen eines Initiators ein,
- die **Akzeptoren** entscheiden über die Kandidatenwerte für die endgültige Entscheidung und
- die **Lernenden** sammeln diese Informationen von den Akzeptoren und melden die endgültige Entscheidung dem Initiator zurück.

- Ein Vorschlag, der von einem Antragsteller gesendet wird, ist ein Tupel (v, n), wobei v ein Wert und n eine Sequenznummer ist.

- Wenn es nur einen Akzeptor gibt, der entscheidet, welcher Wert als Konsenswert gewählt wird, dann wäre diese Situation zu einfach. Was passiert, wenn der Akzeptor abstürzt? Um damit umzugehen, gibt es mehrere Akzeptoren.

- Ein Vorschlag muss von mindestens einem Akzeptor bestätigt werden, bevor er für die endgültige Entscheidung in Frage kommt.

Verteilter Konsens

- Die Sequenznummer wird verwendet, um zwischen aufeinander folgenden Versuchen der Protokollanwendung zu unterscheiden.
- Nach Empfang eines Vorschlags mit einer größeren Sequenznummer von einem gegebenen Prozess, verwerfen Akzeptoren die Vorschläge mit niedrigeren Sequenznummern.
- Schließlich akzeptiert ein Akzeptor die Entscheidung der Mehrheit.

Phasen des Paxos Algorithmus

1. **Die Vorbereitungsphase**
   - Jeder Antragsteller sendet einen Vorschlag (v, n) an jeden Akzeptor
   - Wenn n die größte Sequenznummer eines von einem Akzeptor empfangenen Vorschlags ist, dann sendet er ein *ack* (n, $\perp$, $\perp$) an seinen Vorschlager
   - Hat der Akzeptor einen Vorschlag mit einer Sequenznummer $n' < n$ und einem vorgeschlagenen Wert v akzeptiert, antwortet er mit *ack*(n, v, n').

Verteilter Konsens

2. **Aufforderung zur Annahme eines Eingabewertes**

   - Wenn ein Antragsteller $ack(n, \perp, \perp)$ von einer Mehrheit von Akzeptoren empfängt, sendet er an alle Akzeptoren $accept(v, n)$ und fordert sie auf, diesen Wert zu akzeptieren.
   - Wenn ein Akzeptor in Phase 1 einen $ack(n, v, n')$ an den Antragsteller zurücksendet, muss der Antragsteller den Wert v mit der höchsten Sequenznummer in seiner Anfrage an die Akzeptoren einbeziehen.
   - Ein Akzeptor akzeptiert einen Vorschlag (v, n), sofern er nicht bereits zugesagt hat, Vorschläge mit einer Sequenznummer größer als n zu berücksichtigen.

3. **Die endgültige Entscheidung**

   - Wenn eine Mehrheit der Akzeptoren einen vorgeschlagenen Wert akzeptiert, wird dies der endgültige Entscheidungswert. Die Akzeptoren senden den akzeptierten Wert an die Lernenden, wodurch sie feststellen können, ob ein Vorschlag von einer Mehrheit von Akzeptoren akzeptiert wurde.

Divide-and-Conquer

Replikation

- Replikation dient:
  - Der Gruppenbildung mit Gemeinsamkeiten,
  - Der Vererbung von Verhalten und Spezialisierung
  - Der räumlichen Exploration
  - ..

Beim Divide-and-Conquer" (Teile und herrsche) Ansatz wird versucht ein komplexes Problem soweit rekursiv zu zerlegen dass am Ende nur noch triviale Probeleme übrig bleiben!

- Beispiel: Verteiltes Sensornetzwerk und Bestimmung von geometrisch ausgedehnter Sensoraktivität und Sensorfusion

Verteilter Informationsaustausch

Problem: Wie können Informationen von der Informationsquelle zu Informationssenken, d.h. Agenten die an den Informationen interessiert sind, zugestellt werden?

- Die Replikation kann verwendet werden, um die Zustellungswahrscheinlichkeit zu erhöhen und die Latenz zu verringern.
- Lernende Agenten können die Pfadsuche verbessern, indem sie ihre Reisegeschichte berücksichtigen.
- Datenzentrierte gerichtete Diffusionsalgorithmen können leicht mit autonomen mobilen Agenten implementiert werden.
- *Problem: Flutung des Netzwerks mit Agenten!*

Ereignisbasierte Verteilung

Eine Ereignisbenachrichtigungsalgorithmus kann verwendet werden um effizient eine Kommunikation zwischen Informationsquellen und Senken herzustellen.

- Dazu können Benachrichtigungsagenten verwendet werden die entlang eines Pfades Ereignisse markieren, z.B. dass ein Sensorwert über einer Schwelle liegt.

Verteilter Informationsaustausch

- Suchagenten werden dann benutzt die Ereignisse zu finden die von den Benachrichtigungsagenten hinterlassen wurden.
- Der Ursprung eines Ereignisses kann durch Rückverfolgung des Pfades gefunden werden.

Random Walk

- Eine einfache Möglichkeit besteht darin, dass der Weg zum Empfänger (Datensenke) durch zufällige Richtungswechsel und Migration von datentragenden Agenten erfolgt.
  - Es wird kein geometrisches Modell und Kenntnis des Netzwerkes benötigt

Gerichtete Diffusion

- Durch Replikation der Information bzw. der datentragenden Agenten und grob richtungsorienterte Zustellung mit überlagerten Random Walk und/oder ggf. Backtracking um tote Netzwerkenden (Seitenarme) zu entkommen.
  - Dazu wird partielles geometrisches Modell des Netzwerkes benötigt

Verteilter Informationsaustausch

Beispiel eines Event Agenten in AgentJS

```
function event (dir,target) {
  this.sensors; this.delta={x:0,y:0};
  this.dir=dir; this.target=target;
  this.next = init;

  this.act = {
    init : function () {
      this.delta={x:0,y:0}; this.sensors=[] },
    end : function () { kill() },
    sense : function () {
      rd(['SENSOR',_],function(t){
        this.sensors.push(t[1])
      }) },
    deliver: function () {
     ..
     out(['SENSORS',sensmat]);
     broadcast('node',0,'DELIVER') }
```

Verteilter Informationsaustausch

```
    move : function () {
      switch (this.dir) {
        case DIR.NORTH:
         if (!link(DIR.NORTH)) // edge reached
           this.dir=DIR.WEST,fork({dir:DIR.EAST,target:this.target});
        case DIR.WEST:
         if (!link(DIR.WEST)) // edge reached
           this.dir=DIR.NORTH,fork({dir:DIR.SOUTH,target:this.target,next:move});
        ..
      }
      switch (this.dir) {
        case DIR.NORTH: this.delta.y--; break;
        case DIR.WEST:  this.delta.x--; break;
        ..
      }
      moveto(this.dir) } }

  this.trans = {
    init: sense ,
    sense: function () {
      if ((!exists([this.target])||zero(this.delta))&&this.dir!=DIR.ORIGIN)
        return 'move'; else return 'deliver' },
    move: sense,
    deliver: end }}
```

Verteilter Informationsaustausch

Potentialfeldansatz

- Eine weitere Möglichkeit besteht darin dass dateninteressierte Agenten "farbige" Markierungen in ihrer Umgebung mit Gradienten verteilen
  - Die "Farbe" charakterisiert die Informations/Datenart, z,B. Temperaturssensor, Ortsinformation, usw.
- Ein datentragender Agent wird entlang des Gradienten der Markierungen einen Weg zur Datensenke finden

Verteilte Mustererkennung in Sensornetzwerken

Ausgangssituation: Verteiltes Sensornetzwerk mit Knoten in einem Maschengitternetzwerk.

- Fehlerhafte oder verrauschte Sensoren können Datenverarbeitungsalgorithmen erheblich stören.
- Es ist notwendig, fehlerhafte Sensoren von gut arbeitenden Sensoren zu isolieren.
- Üblicherweise werden Sensorwerte innerhalb eines räumlich nahen Bereichs korreliert, beispielsweise in einem räumlich verteilten mechanischen Lastmonitoringnetzwerks unter Verwendung von Dehnungssensoren.
- Das Ziel des folgenden MAS ist es, ausgedehnte korrelierte Bereiche erhöhter Sensorintensität (im Vergleich zur Nachbarschaft) aufgrund mechanischer Verzerrungen zu finden, die von extern angelegten Lastkräften herrühren.

Verteilte Mustererkennung in Sensornetzwerken

- Es wird ein verteiltes gerichtetes Diffusionsverhalten und eine Selbstorganisation verwendet, die von einem Bildmerkmalsextraktionsansatz abgeleitet sind.
- Es handelt sich hierbei um einem selbstadaptiven Kantendetektor.
- Eine einzelne sporadische Sensoraktivität, die nicht mit der umgebenden Nachbarschaft korreliert ist, sollte von einer erweiterten korrelierten Region unterschieden werden, die das zu erfassende Merkmal darstellt.

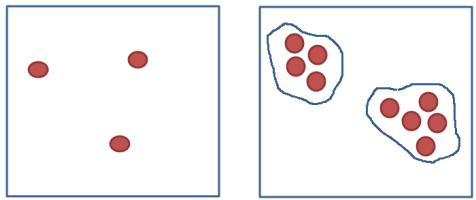


Abb. 5. (a) Nichtkorellierte Sensorstimuli (b) Korrelierte Sensorstimulibereiche

Verteilte Mustererkennung in Sensornetzwerken

Der Algorithmus

Die Merkmals-Erkennung wird vom mobilen Explorationsagenten durchgeführt, der zwei verschiedene Verhaltensweisen unterstützt: Diffusion und Reproduktion.

- Das Diffusionsverhalten wird verwendet, um sich in einem ausgedehnten Bereich zu bewegen, der hauptsächlich durch die Lebensdauer des Agenten begrenzt ist, und um das Merkmal zu detektieren;
    - hier den Bereich mit erhöhter mechanischer Verzerrung (genauer gesagt die Kante eines solchen Bereichs).
- Die Erkennung des Merkmals wird durch das Reproduktionsverhalten verstärkt, das den Agenten veranlasst, am aktuellen Knoten zu bleiben, eine Merkmalsmarkierung zu setzen und mehr Explorationsagenten in der Nachbarschaft auszusenden.

Verteilte Mustererkennung in Sensornetzwerken

- Der lokale Stimulus *H(i,j)* für einen Explorationsagenten, der sich an einem spezifischen Knoten mit der Koordinate (*i,j*) befindet, ist gegeben durch:

$$H(i,j) = \sum_{s=-R}^{R} \sum_{t=-R}^{R} \{|S(i+s, j+t) - S(i,j)| \leqslant \delta\}$$

$$S: \text{ Sensor Signal Strength}$$

$$R: \text{ Square Region around (i,j)}$$

- Die Berechnung von *H* an der aktuellen Position (*i,j*) des Agenten erfordert die Sensorwerte innerhalb des quadratischen Bereichs (der Region von Interesse ROI) *R* um diesen Ort herum.

- Wenn ein Sensorwert *S(i+s,j+t)* mit *i,j* ∈ {*-R*, .., *+R*} ähnlich dem Wert *S* an der aktuellen Position ist (Differenz ist kleiner als der Parameter *d*), wird *H* um eins erhöht.

Verteilte Mustererkennung in Sensornetzwerken

- Wenn der H-Wert innerhalb eines parametrierten Intervalls $D = [e_0, e_1]$ liegt, hat der Explorationsagent das Feature erkannt und verbleibt am aktuellen Knoten, um neue Explorationsagenten zu reproduzieren, die an die Umgebung gesendet werden.
- Wenn H außerhalb dieses Intervalls liegt, wird der Agent zu einem anderen Knoten wechseln und die Exploration (Diffusion) neu starten.
- Die Berechnung von H erfolgt durch eine verteilte Berechnung von Teilsummenausdrücken durch Aussenden von Kind-Agenten an die Nachbarschaft, die selbst mehr Agenten aussenden können, bis die Grenze der Region R erreicht ist.
- Jeder untergeordnete Agent kehrt zu seinem Ursprungsknoten zurück und übergibt den Teilsummenbegriff an seinen übergeordneten Agenten.

Verteilte Mustererkennung in Sensornetzwerken

- Da ein Knoten in der Region R von mehr als einem Kind-Agenten besucht werden kann, setzt der erste Agent, der einen Knoten erreicht, eine Markierung MARK.
  - Wenn ein anderer Agent diese Markierung findet, kehrt er sofort zum übergeordneten Agenten zurück.
- Dieser Mehrwegebesuch hat den Vorteil einer erhöhten Wahrscheinlichkeit, Knoten mit fehlenden (nicht arbeitenden) Kommunikationsverbindungen zu erreichen.
- Ein Eventagent, der von einem Sensingagenten erzeugt wird, liefert schließlich Sensorwerte an Rechenknoten, was hier nicht berücksichtigt wird.

Verteilte Mustererkennung in Sensornetzwerken

Das Agentenverhalten

- Definition von Typen, Körpervariablen, und Hauptklasse Explorer mit den Aktivitäten *init* und *percept*

```
1   κ: { SENSORVALUE,FEATURE, H, MARK }    set of key symbols
2   ξ: { TIMEOUT, WAKEUP }                 set of signals
3   δ: { NORTH,SOUTH, WEST, EAST, ORIGIN } set of directions
4   ε1 =3; ε2 = 6; MAXLIVE = 1;            some constant parameters
5
6   Ψ Explorer: (dir,radius) → {
7     * Body Variables *
8     Σ: { dx, dy, live, h, s0, backdir, group }   global persistent variables
9     σ: { enoughinput, again, die, back, s, v }   local  temporary  variables
10
11    Activities
12    α init: {
13      dx ← 0; dy ← 0; h ← 0; die ← false; group ← ℜ{0..10000};
14      if dir ≠ ORIGIN then
15        ⇔dir; backdir ← ϖ(dir)
16      else
17        live ← MAXLIVE; backdir ← ORIGIN
18      ∀⁺(H,$self,0);
19      ∀⁰(SENSORVALUE,s0?)
20    }
21    α percept: {
22      enoughinput ← 0;
23      ∀{nextdir∈δ | nextdir ≠ backdir ∧ ?Λ(nextdir)} do
24        enoughinput++;
25        Θ→Explorer.child(nextdir,radius)
26      τ⁺(ATMO,TIMEOUT)
27    }
```

Verteilte Mustererkennung in Sensornetzwerken

- Aktivitäten *reproduce* und *diffuse*

```
28    α reproduce: {
29      live--;
30      ∇ˣ(H,$self,?);
31      if ?∇(FEATURE,?) then ∇⁻(FEATURE,n?) else n ← 0;
32      ∇⁺(FEATURE,n+1);
33      if live > 0 then
34       π*(reproduce → init)
35       ∀{nextdir∈δ | nextdir ≠ backdir ∧ ?Λ(nextdir)} do
36         Θ→(nextdir,radius)
37       π*(reproduce → exit)
38    }
39    α diffuse: {
40      live--;
41      ∇ˣ(H,$self,?);
42      if live > 0 then
43       dir ← ℜ{nextdir∈δ | nextdir ≠ backdir ∧ ?Λ(nextdir)}
44      else
45       die ← true
46    }
47    α exit: { ⊗($self) }
48
49    inbound: (nextdir) → {
50      case nextdir of
51      | NORTH →  dy > -radius
52      | SOUTH →  dy < radius
53      | WEST  →  dx > -radius
54      | EAST  →  dx < radius
55    }
56
```

Verteilte Mustererkennung in Sensornetzwerken

- Signalhandler und Hauptübergangsnetzwerk

```
57   Signal handler
58   ξ TIMEOUT: {
59     enoughinput ← 0
60   }
61   ξ WAKEUP: {
62     enoughinput--;
63     if ?∇(H,$self,?) then ∇⁻(H,$self,h?);
64     if enoughinput < 1 then τ⁻(TIMEOUT);
65   }
66
67   Main Transitions
68   Π: {
69     entry → init
70     init → percept
71     percept → reproduce | (h ≥ ε1 ∧ h ≤ ε2) ∧ (enoughinput < 1)
72     percept → diffuse   | (h < ε1 ∨ h > ε2) ∧ (enoughinput < 1)
73     reproduce → exit
74     diffuse → init | die = false
75     diffuse → exit | die = true
76   }
```

Verteilte Mustererkennung in Sensornetzwerken

- Subklasse Kindexplorer

```
77    Explorer child subclass
78    φ child: {
79      α exit      imported from root class
80      ξ TIMEOUT
81      ξ WAKEUP
82      α percept_neighbour {
83        if not ?∇(MARK,group) then
84          back ← false; enoughinput ← 0; ∇ᵗ(MTMO,MARK,group);  ∇ᵡ(SENSORVALUE,s?);
85          h ← (if |s-s0| ≤ DELTA then 1 else 0);
86          ∇⁺(H,$self,h);
87          π*(percept_neighbour → move)
88          ∀{nextdir∈δ | nextdir ≠ backdir ∧ ?Λ(nextdir) ∧ inbound(nextdir)} do
89            Θ→(nextdir,radius)
90          π*(percept_neighbour → goback | enoughinput < 1)
91          τ⁺(ATMO,TIMEOUT)
92      }
93      α move: {
94        backdir ← ϖ(dir); (dx,dy) ← (dx,dy) + ∂(dir);
95        ⇔dir;
96      }
97      α goback: {
98        if ?∇(H,$self,?) then ∇⁻(MARK,$self,h?) else h ← 0;
99        ⇔backdir;
100     }
101     α deliver: {
102       ∇⁻(H,$parent,v?); ∇⁺(H,$parent,v+h);
103       ξWAKEUP ⇒ $parent;
104     }
105     π: {
106       entry → move
107       move → percept_neighbour
108       deliver → exit
109       goback → deliver
110     }
```

Verteilte Mustererkennung und Sensordistribution

- Sensordistribution in verteilten Sensornetzwerken kann strombasiert oder ereignisbasiert erfolgen.

Strombasierte Sensordistribution

Es gibt einen zentralen oder mehrere dezentrale Netzwerkknoten die in periodischen Intervallen die Sensorwerte aller Knoten abfragen - unabhängig davon ob diese sich zu der letzten Anfrage geändert haben

Ereignisbasierte Sensordistribution

Die Sensorknoten liefern Sensordaten zu einem zentralen oder mehreren dezentralen Knoten wenn sich (1) Der Sensorwert geändert hat und (2) es sich um ein ausgedehntes korreliertes Ereignis handelt → Verteilte Mustererkennung

- Sensorwerte können dann per Randomwalk oder gerichteter Diffusion verteilt werden.

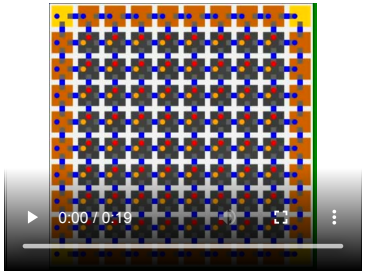
Gerichtete Diffusion

Von einem Quellknoten werden Duplikate von Verteilungsagenten in alle Richtungen ausgesendet, und an den Rändern des Netzwerks zu den Senkeknoten (Rechnern) geleitet.

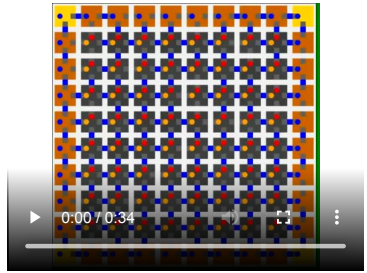
Verteilte Mustererkennung und Sensordistribution

Beispiel in Aktion: Positive Ereigniserkennung

Cluster mit 100% intakten Netzwerkverbindungen



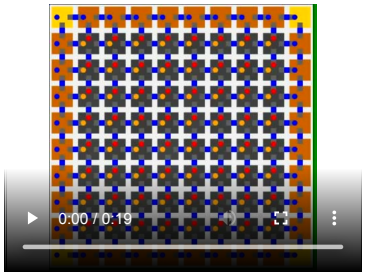
Cluster mit 60% intakten Netzwerkverbindungen



Verteilte Mustererkennung und Sensordistribution

Beispiel in Aktion: Gemischte Situation

Cluster und Störungen



Agenten und Lernen

Maschinelles Lernen

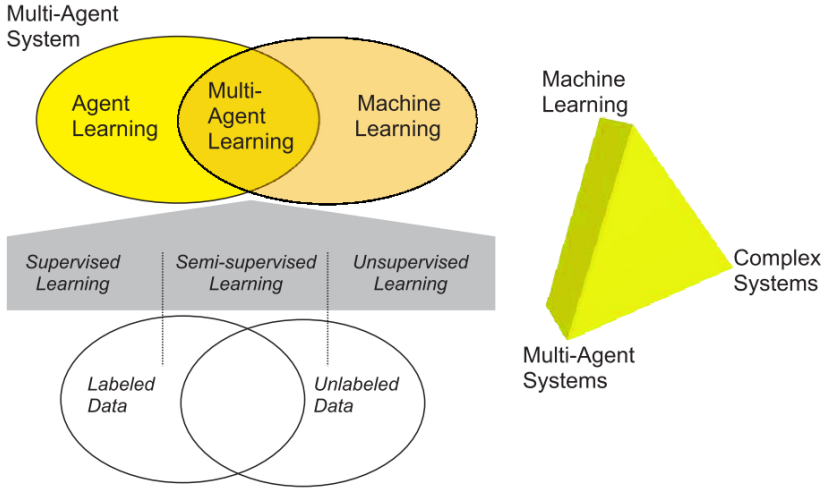


Abb. 6. Maschinelles Lernen lässt sich mit Agenten kombinieren

Rückgekoppeltes Lernen

(Belohnungs- oder Verstärkungslernen)

- Eine sehr populäre maschinelle Lerntechnik zur Lösung von Problemen wird Verstärkungslernen genannt (Sutton and Barto, 1998), eine spezifische Art davon ist bekannt als Q-Learning (Watkins und Dayan, 1992).
- Beim Verstärkungslernen wird abgenommen, dass der Agent in einem Markov-Prozess lebt und in bestimmten Zuständen eine Belohnung erhält.
  - Das Ziel besteht darin, in jedem Zustand die richtigen Maßnahmen zu treffen, um die zukünftige Belohnung des Agenten zu maximieren.
  - Das heißt, die optimale Strategie/Vorgehensweise finden.
- Formal wird das Problem des Verstärkungslernens durch einen Markov-Entscheidungsprozesses (MDP) definiert, in dem die Belohnungen an den Kanten anstatt in den Zuständen angegeben werden

Rückgekoppeltes Lernen

s	$r(s)$
s_1	0
s_2	0
s_3	1
s_4	0

s_i	a	s_j	$T(s_i, a, s_j)$
s_1	a_1	s_1	0.2
s_1	a_1	s_2	0.8
s_1	a_2	s_1	0.2
s_1	a_2	s_4	0.8
s_2	a_2	s_2	0.2
s_2	a_2	s_3	0.8
s_2	a_3	s_2	0.2
s_2	a_3	s_1	0.8
s_3	a_4	s_2	1
s_3	a_3	s_4	1
s_4	a_1	s_4	0.1
s_4	a_1	s_3	0.9
s_4	a_4	s_4	0.2
s_4	a_4	s_1	0.8

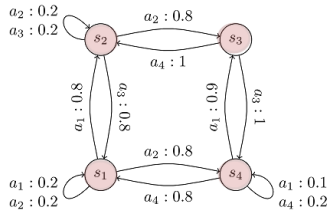
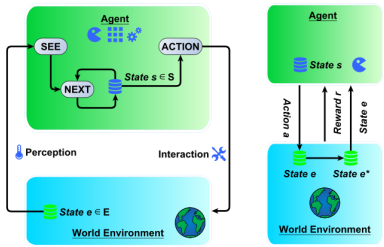


Abb. 7. Grafische Darstellung eines Markov-Entscheidungsprozesses zusammen mit Werten für die Übergangs- und Belohnungsfunktionen. Der Startzustand sei s1, und T die Übergangsfunktion $T(s_i, a, s_j)$ für den Übergang $s_i \rightarrow s_j$.

Rückgekoppeltes Lernen

- D.h. die Belohnungsfunktion ist gegeben durch $r(s_t, a_t) \rightarrow \mathbb{R}$, mit s_t als aktuellen Zustand und a_t als Aktion.
- Es gibt eine Menge von Strategien die Zustände auf Aktionen abbilden, d.h. π:**S** $\rightarrow$ **A**
- *Ein verstärkender Lernagent muss die Strategie* $\pi^{\star} \in \pi$ *finden, die seine zukünftigen Belohnungen maximiert* $\rightarrow$ *optimales Erreichen von Zielen*!

$see : E \rightarrow Per$
$next : I \times Per \rightarrow I$
$action : I \rightarrow Act$
$reward: E \rightarrow A$



Verteiltes Lernen

- Normalerweise werden Lerner zentralisiert, d.h. alle Eingabedaten werden von einem Programm gesammelt und verarbeitet.
- Diese Architektur führt zu einem einzelnen Fehlerpunkt und hohen Datenstromdichten im Netzwerk.
- Es gibt jedoch Ansätze, Lerner mithilfe von Agenten zu verteilen.
- Ein möglicher Ansatz basiert auf der Partitionierung des Lernprozesses in mehreren lokale Lerner, die auf einer räumlichen Datenuntergruppe arbeiten.
- Die lokal gelernten Modelle werden schließlich zu einem globalen Modell fusioniert.
- Dies wird erreicht, indem das (Sensor-) Netzwerk in räumlichen Regionen (Regionen von Interesse ROI) aufgeteilt wird und mehrere Lerner eingesetzt werden, wobei jeder Lerner in einer bestimmten ROI arbeitet.
- Das Lernen von Klassifikationsmodellen und deren Anwendung verwendet daher nur einen lokalen Datensatz, der auch nur eine beschränkte lokale Sicht auf die Welt bietet.

Verteiltes Lernen

- Aus globaler Sicht können die Ergebnisse mehrerer lokaler Klassifikationen abweichen.
- Eine geeignete Methode zur Ableitung einer zuverlässigen globalen Klassifikation (Aufbau des globalen Modells) kann durch einen Mehrheitswahl- und einem Wahlprozess umgesetzt werden.
- Jeder lokale Lernende wählt eine Klassifikationsvorhersage.
    - Die Annahme ist, dass die Mehrheitsentscheidung das wahrscheinlichste Ergebnis liefert.
- Die verteilten Lernersysteme haben im Vergleich zum zentralen Lerner eine sehr gute Skalierungsfähigkeit, und es gibt keinen einzigen Fehlerpunkt.
    - Defekte Knoten oder fehlende Stimmen verringern nur die globale Vorhersagegenauigkeit.

Verteiltes Lernen

$$
\begin{array}{c}
M : D \rightarrow h(S) \\
l \in \mathbf{L} \\
h : S \rightarrow l \\
D : \{(S^1, l^1), (S^2, l^2), \cdots\} \\
S : \begin{pmatrix} x_{1,1} & \cdots & x_{n,1} \\ \vdots & \ddots & \vdots \\ x_{1,m} & \cdots & x_{n,m} \end{pmatrix}
\end{array}
\xrightarrow{\text{Distribution}}
\begin{array}{c}
m_{i,j} : d_{i,j} \rightarrow h_{i,j}(s) \\
h_{i,j} : s_{i,j} \rightarrow l_{i,j} \\
K : (l_{1,1}, l_{1,2}, \cdots) \rightarrow l \\
d_{i,j} : \{(s^1_{i,j}, l^1), (s^2_{i,j}, l^2), \cdots\} \\
s_{i,j} : \begin{pmatrix} x_{i-u,j-v} & \cdots & x_{i+u,j-v} \\ \vdots & \ddots & \vdots \\ x_{i+u,j-v} & \cdots & x_{i+u,j+v} \end{pmatrix}
\end{array}
$$

- M: Zentraler Lerner
- D: Globale Trainingsdatensätze
- h: Globales Modell
- S: Globale Sensordaten
- l: Labels (Klassenattribute)

- k: Lokaler Lerner
- d: Lokale Trainingsdatensätze
- m: Lokales Modell
- s: Lokale Sensordaten
- K: Globaler Aggregator

Verteiltes Lernen

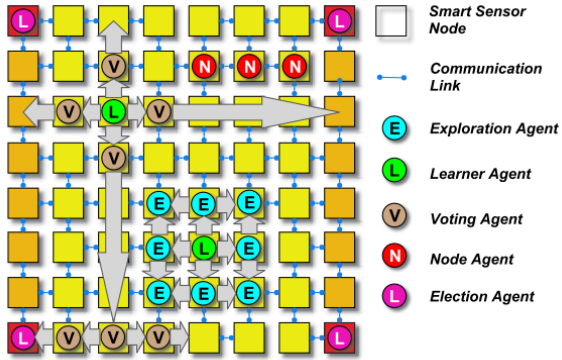


Abb. 8. Logische Netzwerkansicht und Population mit mehreren Agenten, die aus verschiedenen Verhaltensklassen instantiiert wurden.

Verteiltes Lernen

Multiagenten System

Knotenagent

- Jeder Sensorknoten ist mit einem nicht mobilen Knotenagenten besetzt, der Sensorakquisition, Sensorvorverarbeitung und Ereignisdetektion durchführt (d.h. einen signifikanten Stimulus erkennt).
- Ein Knotenagent kann neue Agenten für bestimmte Aufgaben instantiieren oder bereits aktive Agenten benachrichtigen.
- Der Knotenagent ist stationär (nicht mobil) und hat erweiterte Rechte (Systemrolle).

Verteiltes Lernen

Lerner

- Jeder Sensorknoten hat mindestens einen Lerneragenten, der vom Knotenagenten instantiiert und aktiviert wird, wenn ein Sensorstimulus erkannt wurde.
- Dieser Lerner hat zwei verschiedene Modi: (I) Lernen (II) Klassifizierung mit einem gelernten Modell.
- Die Modusauswahl wird von Benachrichtigungsagenten durchgeführt, die im Netzwerk verteilt sind, und
  A. Die Lerner über eine charakteristische Belastungssituation informieren (und ein Label *l* bereitstellen) und die Lerner veranlassen, einen Trainingsdatensatz mit einem spezifischen Label zu erstellen, II. Lerner umschalten in den Anwendungsmodus.
- Die Lerneragenten haben Zugriff auf Sensordaten aus der nahen Umgebung, die von Exploreragenten gesammelt und über die Tuple-Space-Datenbank (einem Plattformdienst) weitergeleitet werden.
- Die einzelnen Lerner erstellen ein lokales Sensor-Last-Vorhersagemodell.

Verteiltes Lernen

Explorationsagent

- Dieser Agent liefert Input für die Vorhersage eines signifikanten Sensorstimulus und für das Lernen die Sensordaten in einer räumlich eingeschränkten Region of Interest (ROI).
- Die räumliche Sensorexploration wird mit einem Divide-and-Conquer-Ansatz durch eine Gruppe von Explorationsagenten durchgeführt, die die Sensordaten sammeln und die Daten an die Knoten- oder Lerneragenten liefern.
- Jeder Explorerationsagent, der auf einem bestimmten Knoten in der ROI arbeitet, erstellt Explorationskindagenten, die Daten auf Nachbarknoten untersuchen.

Verteiltes Lernen

Wahl- und Abstommungsagent

- Wenn ein Lerneragent eine Lastsituation aus seiner lokalen Sicht klassifiziert (und das lokale Modell lokale Daten verwendet), sendet er Abstimmungsagenten mit einer Vorhersage der Lastsituation.
- Die Abstimmungsagenten übermitteln die Stimmen an Wahlagenten, die eine Mehrheitswahl für eine globale und wahrscheinlichste Vorhersage einer Lastsituation durchführen.
- Die meisten Agenten werden dynamisch von anderen Agenten erstellt, z. B. werden die Explorationsagenten von Knoten, Lernern und anderen Explorer-Agenten erstellt.
- Die Agenteninteraktion erfolgt über Tuple-Spaces (synchronisierter Datenaustausch basierend auf Patterns). Darüber hinaus werden mobile Signale zur Benachrichtigung anderer Agenten verwendet.

Verteiltes Lernen

Use Case: Strukturüberwachung

- Zu Beginn unbekannte äußere Kräfte, die auf eine mechanische Struktur einwirken, führen zu einer Verformung des Materials aufgrund der inneren Kräfte.
- Ein materialintegriertes aktives Sensornetzwerk bestehend aus Sensoren, Elektronik, Datenverarbeitung und Kommunikation kann zusammen mit mobilen Agenten verwendet werden, um relevante Sensoränderungen mit einem ereignisbasierten Informationsverteilundsverhalten zu überwachen.
- Inverse numerische Methoden können schließlich die Materialantwort berechnen. Die Antwort des unbekannten Systems für die extern angelegte Last *I* wird durch die Dehnungssensor-Stimulationsantwort *s*' (eine Funktion von *s*) gemessen, und schließlich berechnen die inversen numerischen Verfahren eine Approximation *I*' der angelegten Last.
- Neben komplexen numerischen Verfahren kann verteiltes Maschinelles Lernen für eine schnelle und effiziente Bestimmung bestimmter ausgewiesener Lastfälle verwendet werden.

Verteiltes Lernen

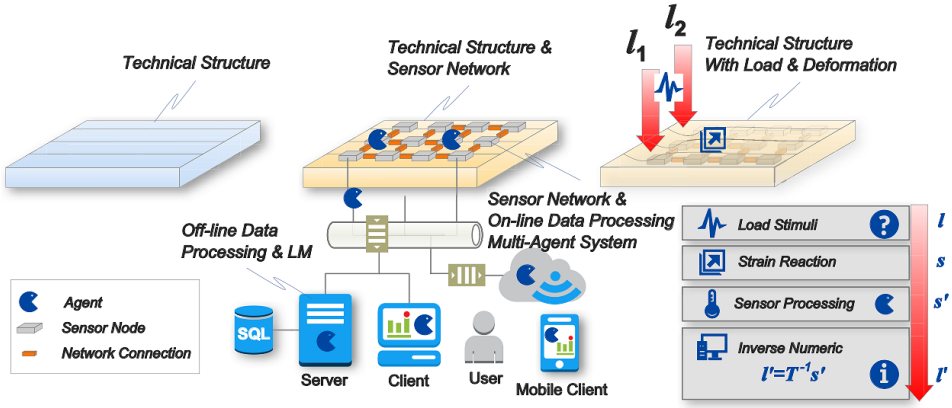


Abb. 9. Vom Material zur intelligenten überwachten Struktur mit mobilen Agenten

Verteiltes Lernen

Use Case: Erdbebenüberwachung und Crowd Sensing

- Verteiltes agentenbasiertes Lernen wird verwendet um aus seismischen Sensordaten auf verschiedene Erdbebenereignisse zu schließen

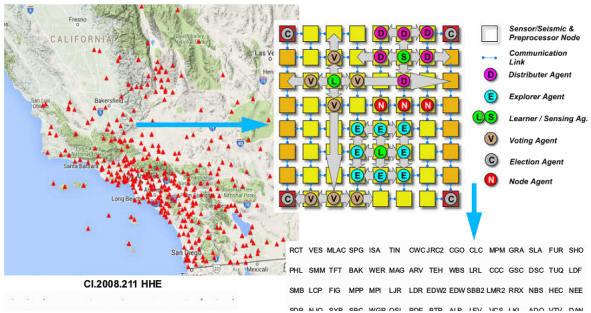


Abb. 10. (Oben, links): Das südkalifornische seismische Sensornetzwerk [Google Maps] (Oben, rechtes) Sensornetzwerk mit Stationen, die auf einer logischen zweidimensionalen Mesh-Grid-Topologie mit räumlicher Nachbarschaftsplazierung abgebildet wurden und Beispielpopulation mit verschiedenen mobilen und immobilen Agenten [1]

Verteiltes Inkrementelles Lernen

- Neben dem grob granulierten Zyklus **Lernen → Modell → Klassifikation** mit einer Datenmenge (Trainingsdaten) **D**=$\{D_1,D_2,..,D_n\}$ kann das Lernen auch inkrementell erfolgen (strombasiertes Lernen)
- Dabei wird das Modell schrittweise mit neuen Datensätzen aufgebaut → schwierig je nach Algorithmus und Modellklasse (z.B. Entscheidungsbaum)
  - Entscheidungsbäume bestehen aus Knoten die Eingabevariablen nutzen um die Klassifikation optimal durch Pfaditeration zu erreichen
  - Welche Variablen optimal sind (Merkmalsselektion) wird von den Trainingsdaten bestimmt
  - Sind diese nur unvollständig bekannt, kann eine nachträgliche Erweiterung des Baumes zur Verwendung ungeeigneter (schwacher) Variablen führen → Schlechte Klassifikationsergebnisse sind die Folge!

Verteiltes Inkrementelles Lernen

Rückkopplung

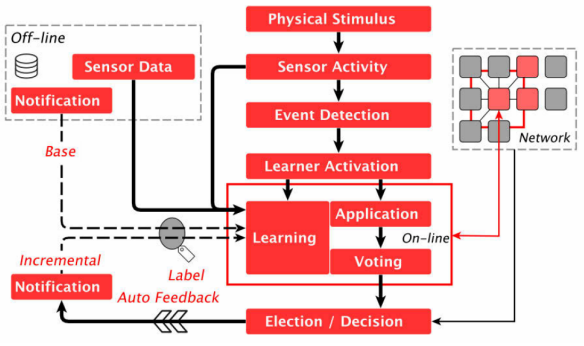


Abb. 11. Das Grundkonzept: Globales Wissen basierend auf Mehrheitsentscheidungen wird an lokale Lerninstanzen zurückgegeben, um das erlernte Modell zu aktualisieren.