Self-stabilizing distributed data fusion

B. Ducourthial, V. Cherfaoui, and T. Denoeux

Lab. Heudiasyc UMR CNRS-UTC 7253
Université de Technologie de Compiègne
France
Corresponding author Bertrand.Ducourthial@utc.fr

Abstract. The Theory of Belief Functions is a formal framework for reasoning with uncertainty that is well suited for representing unreliable information and weak states of knowledge. In information fusion applications, it is mainly used in a centralized way, by gathering the data on a single node before computation.
In this paper, a distributed algorithm is proposed to compute the neighborhood confidence of each node, by combining all the data of its neighbors using an adaptation of the well known Dempster's rule. Moreover, a distributed algorithm is proposed to compute the distributed confidence of each node, by combining all the data of the network using an adaptation of the cautious operator. Then, it is shown that when adding a discounting to the cautious operator, it becomes an r-operator and the distributed algorithm becomes self-stabilizing. This means that it converges in finite time despite transient faults.
Using this approach, uncertain and imprecise distributed data can be processed over a network without gathering them on a central node, even on a network subject to failures, saving important computing and networking resources. Moreover, our algorithms converge in finite time whatever is the initialization of the system and for any unknown topology.
This contribution leads to new interesting distributed applications dealing with uncertain and imprecise data. This is illustrated in the paper: an application for sensors networks is detailed all along the paper to ease the understanding of the formal approach and to show its interest.

1 Introduction

Algorithms for gathering data spread out over a network of communicating process units are well known [17, 25, 7]. However, in the real world, information is almost always tainted with various kinds of imperfection, such as imprecision, uncertainty, ambiguity, etc. Following [10], if a variable X takes its values in Ω (domain or *frame of discernment*), an item of information about X could be represented as a pair $(value, confidence)$. The *value* component corresponds to a subset of Ω while the *confidence* component is an indication on the reliability of the item of information. Imprecision is related to the *value*, uncertainty is related to the *confidence*. For instance, when using the output of any disposal (sensor, algorithm, model, expert...), it would be preferable to distinguish between the following pieces of information: "the value is between 15 and 25", "the value is probably 20", "the value is probably between 15 and 25". The first one is imprecise but certain, the second is precise but uncertain while the last is both imprecise and uncertain. The Set-Membership approach can represent the imprecision but lacks robustness while the Probability theory models aleatory uncertainty but does not express any notion of imprecision. The Theory of Belief Functions has been introduced by Dempster (1968) [5] and Shafer (1976) [18], and has been further developed by Smets (Transferable Belief Model) in the 1990's [23]. It is also known as Dempster-Shafer theory or Evidence. It is a formal framework for representing and reasoning from partial (uncertain, imprecise) information, by generalizing both the Set-Membership approach and the Probability Theory. Many applications in the field of data fusion are developed through belief functions framework [21]. However, even if the sources of data are distributed in space or in time, the proposed approaches are variant of centralized fusion methods [16].

As more and more sensors are present in our life (in smart-phones, vehicles, clothes, body, etc.), and as more and more networking connections appear between all these devices, a distributed approach for computing belief functions appear useful and is promising to many applications. In fact, such an approach would not be limited to information produced by sensors but could be applied to *any* imprecise and uncertain information, even on the distributed system itself. Recent works have been done in [3, 2] where each node discounts information according to the distance and the age of the received message before to combine it with a local knowledge. The notion a data contamination due to the vehicular network context was taken into account for the choice of the combination rules. This work has been extended in [26]. In [15], a spanning tree is used for dealing with the loops of the network. In all these works, the network is supposed to be reliable.

Instead of gathering the information and then processing it in a central node, it would be very advantageous in terms of networking and computing resources to compute locally the belief functions. Generally, every node produces locally an information and then a local belief function (called in the following *direct confidence*). It would be very interesting to enrich such a confidence with information from other nodes. However in many cases, the result will depend on the

position of the node and it is expected that node u should have a different result as compared to that of node v. In this (very common) case, computing the belief function locally using a distributed algorithm appears to be the best approach.

Nevertheless, distributed algorithms are subject to faults, especially when the devices are cheap and the underlying network opportunistic. We present in this paper algorithms able to compute a belief function on every node of a network subject to crash and transient faults. Our first algorithm computes on every node its *neighborhood confidence* relying on the direct confidences of neighbors. The second algorithm builds on every node its *distributed confidence*, taken into account all the direct confidences produced in the network, while favoring the closest ones. Our algorithms are self-stabilizing, so that they recover correct behavior after finite time starting from an arbitrary global state caused by a transient fault [8, 9]. All these results are given for a simplified communication model relying on a simple `push` action, periodically called by the nodes. This can be implemented in an idealized WiFi network or in the classical shared-register model. The correctness of the algorithms is shown thanks to previous works on *r-operators* [12]. By modeling local algorithms with operators, global properties (termination, self-stabilization in different communication model) can be inferred by checking the algebraic properties of the operators [14, 13, 4]. However, for applying such a general scheme (and reusing generic proofs), the problem to be solved has to be modeled as an algebraic operator.

The contributions of our paper are threefold. First we explain how the processing of uncertain and imprecise data in a distributed system can be modeled by algebraic operators over a specific finite set, namely *vectors of discretized weights*. Second we propose two distributed algorithms for computing data fusion over distributed data in a network of unknown topology, the first one combining close information, the second one combining also remote information. Finally, we show that this second algorithm can be modeled as an r-operator (namely *discounted cautious* over the vectors of discretized weights), that satisfies the requirements for ensuring the self-stabilization of the distributed system.

Such contributions allows to process uncertain and imprecise distributed data without gathering them on a central node, even on a network subject to failures, saving important computing and networking resources. Moreover, our algorithms converge in finite time whatever is the initialization of the system and for any unknown topology. We believe that many applications can take benefit of this approach; we detail an application for sensors networks all along the paper to ease the understanding of the formal approach and to show its interest.

In Section 2, we present the distributed system we consider. Then, in Section 3, we explain how to model the processing of uncertain and imprecise data using local computations based on an adaptation of the Dempster'rule over a specific set (the vectors of discretized weights), and we present an algorithm for neighborhood confidence computation. In Section 4, we extend these results by presenting a distributed algorithm able to process all the uncertain and imprecise data of the distributed system. We show that such algorithm can be modeled as an r-operator (discounted cautious) and is self-stabilizing.

2 Self-stabilizing Distributed Systems

System. We consider a distributed system $\mathcal{S}$ composed of communicating computing nodes. Each node owns a local memory and a sequential computing unit so that it is able to run a local algorithm. Nodes are not synchronized. The local memory of node v is composed by its *private memory* $\texttt{PRIV}_v$, an *incoming memory* $\texttt{IN}_v$ and an *output memory* $\texttt{OUT}_v$. The private memory of v contains its direct confidence and is regularly updated thanks to an external local disposal (eg. a sensor). The output memory will store the result of the local computation on v, namely its *neighborhood confidence* (Algorithm 1, Section 3) or its *distributed confidence* (Algorithm 2, Section 4). Communications are done through a simple atomic action called `push`: when a *sender* node u executes `push(m)`, the value m stored in its output memory is copied into the input memories of some *receiver* nodes v_1, v_2, ..., v_k.

We assume transient faults sometimes occur at the memories. To circumvent this problem, we will introduce *self-stabilization.*

Moving topology. The receivers of a `push` action on v are not known from the sender v and do not know v. They are determined by the current topology of $\mathcal{S}$ and could be different from those of a previous `push` on the same node v. There is a *link* (u, v) between u and v if a data m pushed by u is received by v. Such a link disappears when a data m' pushed by u is not received by v. A link (u, v) may exist while the link (v, u) does not exist. The channel capacity is a single message.

In order our algorithms stabilizes, it is required that the topology remains stable for a period longer than the stabilization phase. We say that the topology of $\mathcal{S}$ *stabilizes* if it remains the same for further `push` actions (same links, that is, same receivers for a given sender). Such a topology is modeled by a directed graph $G(V, E)$ where V is the set of nodes and E is the set of current links. We denote by Γ_v^{01} the set of ancestors of v included v itself: $\Gamma_v^{01} = \{v\} \cup \{u \in V, (u, v) \in E\}$ and by Γ_v the set of all ancestors of v included v itself: $\Gamma_v = \{v\} \cup \{u \in V, \exists u_1, \ldots, u_k \in V \; s.t. \; (u, u_1), (u_1, u_2), \ldots, (u_k, v) \in E\}$.

Example. This communication scheme can be implemented on a wireless network with a link capacity of a single message: a `push` is implemented using a local broadcast followed by an idle period longer than the maximal communication duration (which is bounded in wireless protocols such as IEEE 802.11). Nodes moves and collisions add/delete links according to the communication range.

When the topology remains stable, this communication model can also be implemented through shared registers: a `push` by a writer u is simply a write into the register it shares with some readers v_1, ..., v_n. Then transformers can be used to extend this model to other communication models [1, 11].

In the rest of this paper, we develop an example in the context of wireless sensor networks, where each node regularly push its result to potential neighbors. Nodes only own a local clock and may push their results at different frequencies.

Self-stabilization. A *configuration* of a distributed system $\mathcal{S}$ is an instance of the states of its processors and links. The set of configurations of $\mathcal{S}$ is denoted as $\mathcal{C}$. A distributed algorithm is a collection of local algorithms running on every node of $\mathcal{S}$. Processors actions change the global system configuration. An *execution* e is a sequence of configurations $c_1, c_2, \ldots$. Configuration c_1 is the *initial configuration* of execution e.

A *specification* is a predicate on executions that are admissible for a distributed system. A system *matches its specification* if all its possible executions match the specification. This paper considers problems whose solutions consist in computing a global result (static task); the specification can then be given in terms of a set of configurations. The set of configurations that matches the specification of static problems is called the set of *legitimate* configurations (denoted as $\mathcal{L}$).

Self-stabilization is defined through the concept of closed attractor.

Definition 1 (Closed Attractor). *Let $\mathcal{C}_a$ and $\mathcal{C}_b$ be subsets of $\mathcal{C}$. $\mathcal{C}_b$ is an* attractor *for $\mathcal{C}_a$ if and only if for any initial configuration $c_1 \in \mathcal{C}_a$, for any execution $e = c_1, c_2, \ldots$, there exists $i \geq 1$ such that $c_i \in \mathcal{C}_b$. It is* closed *if for any $j \geq i$, $c_j \in \mathcal{C}_b$.*

In the usual (non-stabilizing) distributed systems, possible executions can be restricted by allowing the system to start only from some well-defined initial configurations. In stabilizing systems, problems cannot be solved using this convenience, since all possible configurations are admissible initial configurations.

Definition 2 (Self-stabilization). *A system $\mathcal{S}$ is called self-stabilizing if and only if there exists a non-empty subset $\mathcal{L} \subset \mathcal{C}$ of legitimate configurations such that $\mathcal{L}$ is a closed attractor for $\mathcal{C}$.*

3 Neighborhood Confidence Algorithm

In this section, we consider a network where each node owns a private data and we propose a distributed algorithm for computing a neighborhood confidence. After summarizing our approach, we explain how to build the domain of our variables (*vectors of discretized weights*). Next we introduce an adaptation of the Dempster operator for data combination. We then present our algorithm and its properties. We terminate by explaining how to exploit its outputs.

3.1 Neighborhood confidence principle

We consider a network where each node owns a private data. Such an information is regularly updated using a local external disposal (sensor, other algorithm...). As the data are uncertain and imprecise, instead of collecting on each node the data of its neighbors, the purpose of our algorithm is to evaluate a *neighborhood confidence* using the *direct confidences*, these lasts being computed by each node starting from their private data and their local external disposal.

Our scheme is general enough for covering many applications but to fix ideas, we illustrate it all along the paper using an example (marked with a vertical rule): a very simple weather forecast application. We assume that each node is able to measure the local atmospheric pressure and to determine whether it is decreasing, stable or increasing, allowing us to deduce a weather forecast. As the accuracy of the measurement is not perfect, we consider intervals instead of reals: each pressure measurement is an interval $I \subset \mathbb{R}^+$ and the pressure gradient ΔI computed with the two last measures I_k and I_{k-1} is then an interval of $\mathbb{R}$. Moreover, the sensor is not totally trusted because it could be damaged. Hence, we consider a confidence in the information, by affecting so-called *masses* to the sets ΔI and $\mathbb{R}$ in such a way that the sums of the masses is 1. The mass on $\mathbb{R}$ corresponds to the proportion of time when the sensor is not working correctly; the more the sensor is reliable, the lower is the mass on $\mathbb{R}$. Hence, thanks to its disposal, each node v obtains a result which can be interpreted as follows: "I have a confidence of 80% that the atmospheric pressure is increasing or stable, announcing a good weather". This is the *direct confidence* of node v.

The direct confidences are the local inputs of our algorithms. Starting from them, our first algorithm builds on each node v its *Neighborhood Confidence* by combining the direct confidence of v with those it receives from its direct ancestors. For this purpose, the confidences are stored as *vectors of discretized weights*; they are combined using an adaptation of the Dempster'rule.

3.2 Domain $\mathbb{K}$: vectors of discretized weights

In this section, the domain on which operate our algorithms is introduced.

The state of belief of a node is expressed on a *frame of discernment* Θ using a *basic belief assignment* (BBA for short). Such a BBA can be represented by several means, the most common one being with a *mass function*. A mass function m^Θ is a mapping from the set of subsets of Θ, denoted $\mathcal{P}(\Theta)$, to the set of *masses* $[0, 1] \subset \mathbb{R}$ such that $\sum_{X \subset \Theta} m^\Theta(X) = 1$. A set $X \subset \Theta$ such that $m(X) > 0$ is called *focal set*. If every focal set X satisfies $|X| = 1$, m is said to be Bayesian and it corresponds to a probability mass function. However, the main interest of the Theory of Belief Functions is to consider every subset X of Θ. The more a node is confident in X, the higher is $m^\Theta(X)$. If the empty set $\emptyset$ is not a focal set, the mass is *normal*. A mass on $\emptyset$ is used to model conflict between pieces of evidence on which m is based. If Θ is not a focal set, the mass is *dogmatic*. A mass on Θ is used to model lack of knowledge. The higher $m^\Theta(\Theta)$ is, the less the mass function m^Θ is informative. If $m^\Theta(\Theta) = 1$, the mass function is *vacuous*. Finally, a mass function is *simple* if it admits at most two focal sets including Θ.

In our example, the pressure gradient interval ΔI belongs to $\Theta = \mathbb{R}$. Each node then determines a simple mass function m^Θ such that $m^\Theta(\Delta I) = 1 - \alpha$ and $m^\Theta(\Theta) = \alpha$. The size of pressure measure interval (and then the size of ΔI) is related to the accuracy of the measure, while α is related to the reliability of the measure disposal (sensor).

Starting from a mass function m^Θ on the frame of discernment Θ, it is convenient to build another mass function on a coarser, finite frame of discernment Ω. Such a *coarsening* allows us to work on a finite set with simple interpretation. It will also limit the amount of data exchanged between nodes.

For our simple weather forecast example, we consider $\Omega = \{wet, cloud, sun\}$. Each node determines a simple mass function m^Ω depending on the position of $\Delta I \in \mathbb{R}$ regarding 0. If $\Delta I << 0$ (case `a` in Fig. 1), then $m^\Omega(\{\text{wet}\}) = 1 - \alpha$ while if $\Delta I >> 0$ (case `d`), then $m^\Omega(\{\text{sun}\}) = 1 - \alpha$. When ΔI is close to 0, there are some uncertainties: if $0 \in \Delta I$ (case `e` in Fig. 1), then $m^\Omega(\Omega) = 1$ because a node cannot determine whether the pressure increase or not; if $0 < \Delta I$ (case `c`) then $m^\Omega(\{\text{cloud}, \text{sun}\}) = 1 - \alpha$, while if $\Delta I < 0$ (case `b`), then $m^\Omega(\{\text{wet}, \text{cloud}\}) = 1 - \alpha$.

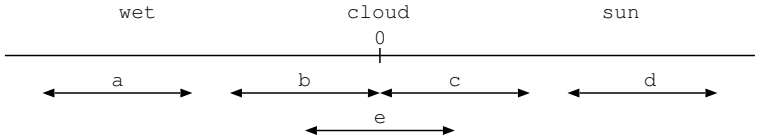

Fig. 1. Determining m^Ω from the comparison of ΔI with 0.

Besides classical mass functions, a basic belief assignment can be represented by other functions, such as commonality and weights functions. Our algorithms work with weights, which are obtained from masses using commonalities [20] [6], as summarized in the following table.

mass function	commonality function	weight function
$m : \mathcal{P}(\Omega) \to [0,1]$	$q : \mathcal{P}(\Omega) \to [0,1]$	$\mu : \mathcal{P}(\Omega) \setminus \Omega \to \mathbb{R}^+$
$A \mapsto m(A)$	$A \mapsto q(A)$	$A \mapsto w(A)$
$\sum_{A \subset \Omega} m(A) = 1$	$q(A) = \sum_{B \subset \Omega, A \subseteq B} m(B)$	$\mu(A) = \Pi_{B \subset \Omega, A \subseteq B} q(B)^{(-1)^{\lvert B \rvert - \lvert A \rvert + 1}}$

In our example, we considered simple mass functions m^Ω defined by $m^\Omega(X) = 1 - \alpha$ for a single subset $X \subset \Omega$ and $m^\Omega(\Omega) = \alpha$. We then obtain $q^\Omega(\emptyset) = q^\Omega(X) = 1$ and $q^\Omega(Y) = \alpha$ for any other subset Y of Ω not included in X. Regarding the weight functions, we obtain $\mu^\Omega(X) = \alpha$ and $\mu^\Omega(Y) = 1$ for any other $Y \subsetneq \Omega$ (our approach works also with more complex mass functions).

Whenever the mass functions are not dogmatic ($m(\Omega) > 0$), the weights are strictly positive. Moreover, any separable mass function ensures that the weights are smaller than or equal to 1 and reciprocally. A mass function m is *separable* ([18] Chapter 4) if it admits a canonical decomposition in simple mass functions m_i so that the conjunctive combination of these simple mass functions is equal to the mass function itself: $m = \bigcirc\!\!\!\!\cap m_i$. We introduce the conjunctive operator $\cap$ hereafter. Hence, by restricting the considered mass functions to the set of separable non dogmatic normalized mass functions, we can represent them as weight functions from $\mathcal{P}(\Omega) \setminus \{\Omega, \emptyset\}$ to the interval $(0,1] \subset \mathbb{R}$. The data set we consider is then a set of values in $(0,1]$, one per subset of Ω except $\emptyset$ and Ω.

However, to ensure convergence in finite time, finite memory consumption and finite message size, we consider a discretization of $(0,1]$, denoted by $\mathbb{W}$:

$\mathbb{W} \subset (0,1]$ with $|\mathbb{W}| \in \mathbb{N}$ and $1 \in \mathbb{W}$. We denote by $\epsilon \in \mathbb{W}$ the smallest element of $\mathbb{W}$.

As a conclusion, the data set of our algorithms is $\mathbb{W}^{2^{|\Omega|}-2}$, that is vectors of $2^{|\Omega|}-2$ values taken into $\mathbb{W}$, which is a discretization of the weights that represent a BBA expressing a state of knowledge over a frame of discernment Ω. We call this set *vectors of discretized weights* and we denote it $\mathbb{K}$. Any vector of weights $\mathbf{w}$ in $\mathbb{K}$ can be coded with $(2^{|\Omega|}-2)\ln(|\mathbb{W}|)$ bits. We denote by $\mathbf{w}_\perp$ (resp. $\mathbf{w}_\top$) the element of $\mathbb{K}$ composed only with weights ϵ (resp. 1).

In our example, supposing we discretize $(0,1]$ up to the thousandth, as $|\Omega| = 3$, the vectors of weights require a size of 60 bits.

3.3 Operations on $\mathbb{K}$: discretized Dempster operator

The BBAs can be combined using some operators in the aim of forging a better knowledge from several sources of information. Given two mass functions m_1 and m_2 over the same discernment set Ω, the *conjunctive operator* $\textcircled{\cap}$ builds a new mass function denoted $m_{1\textcircled{\cap}2}$ by emphasizing the agreement between the sources that induced the BBAs, providing they are reliable [19]. The sources should be independent, that is, they provide distinct, non overlapping pieces of evidence [18]. The conflict between two BBAs m_1 and m_2 is given by $m_{1\textcircled{\cap}2}(\emptyset)$. It can be spread over other sets when the conflict is ignored. The resulting operator is called *Dempster*'s rule, denoted by $\oplus$. Operators $\textcircled{\cap}$ and $\oplus$ are commutative and associative and admit the vacuous mass function as neutral element.

Conjunctive operator	Dempster operator
$m_{1\textcircled{\cap}2}(A) = \sum_{B\cap C=A} m_1(B)\cdot m_2(B)$	$m_{1\oplus 2}(A) = \begin{cases} m_{1\textcircled{\cap}2}(A)/\left(1-m_{1\textcircled{\cap}2}(\emptyset)\right) & A \neq \emptyset \\ 0 & A = \emptyset \end{cases}$

In our example, supposes that node u determines its direct confidence as a mass function m_{du} such that $m_{du}(\{\text{cloud}, \text{sun}\}) = 0.8$ and $m_{du}(\Omega) = 0.2$ (hence its disposal is reliable at 80% but the pressure gradient ΔI was close above 0). Supposes that a neighbor v of u determines its direct confidence as a mass function m_{dv} such that $m_{dv}(\{\text{sun}\}) = 0.7$ and $m_{dv}(\Omega) = 0.3$ (v trusts its disposal at 70% only but the pressure gradient is clearly above 0). Then, by combining these two BBAs, we find: $m_{du\oplus dv}(\{\text{sun}\}) = 0.7$. Now, if another neighbor w determines its direct confidence as a mass function m_{dw} such that $m_{dw}(\{\text{wet}, \text{cloud}\}) = 0.9$ and $m_{dw}(\Omega) = 0.1$, the belief in "sun" decreases to 0.538, but is not null because w does not fully trust its disposal.

When the BBAs are expressed with weight functions, the Dempster operator becomes a product: if μ_1 and μ_2 are two weight functions expressing BBAs on the same discernment frame Ω, then $\mu_{1\oplus 2} = \mu_1 \oplus \mu_2$ is defined by: $\mu_{1\oplus 2}(A) = \mu_1(A) \times \mu_2(A)$. Nevertheless, as the weights we manipulate belong to the finite set $\mathbb{W}$, we need to introduce a product operation on $\mathbb{W}$, that we denote by $*$. We then obtain a discretized Dempster-like operator denoted $\boxplus$ on $\mathbb{K}$ as follows. For any vector $\mathbf{w}_1$ and $\mathbf{w}_2$ belonging to $\mathbb{K}$, $\mathbf{w}_{1\boxplus 2} = \mathbf{w}_1 \boxplus \mathbf{w}_2$ is defined by: $\mathbf{w}_{1\boxplus 2}(A) = \mathbf{w}_1(A) * \mathbf{w}_2(A)$ for any subset A of Ω except Ω and $\emptyset$. By lack of place, the "discrete multiplication" $*$ is not defined here.

As a conclusion, in our first algorithm, the direct confidence expressed as vectors of $\mathbb{K}$ will be combined using the operator $\boxplus$, an adaptation of the Dempster operator using the operator $*$ for multiplying the weights of $\mathbb{W}$.

3.4 Algorithm 1: Neighborhood Confidence computation

Now that we have defined the data set and the operator used to combine the data, the algorithm is simply described as follows. The direct confidence of each node is regularly updated by an external mean, as explained previously, and stored in the private memory $\texttt{PRIV}_v$. It is coded (as all other variables) by a vector of discretized weights belonging to the finite set $\mathbb{K}$. The incoming memory $\texttt{IN}_v$ on node v stores all data pushed by some ancestors since the last timer expiration. The output memory $\texttt{OUT}_v$ contains the neighborhood confidence computed by v.

Nodes are not synchronized. Timers are given by local clocks and may have an unbounded drift. Upon timer expiration, each node computes its neighborhood confidence by combining its own direct confidence with those it has received since the last timer expiration, using operator $\boxplus$. It also pushes its direct confidence.

Algorithm 1: **Neighborhood Confidence, node** v

```
1   Upon timer expiration:
2     PRIV_v ← current direct confidence
3     OUT_v ← PRIV_v                    ▷ Initializing the iterative computation
4     for each entry u in IN_v do       ▷ Iterative computation of the output
5       OUT_v ← OUT_v ⊞ IN_v[u]
6     end for
7     push( PRIV_v )                    ▷ Sending the direct confidence to neighbors
8     Reset IN_v
9     Restart the timer
```

The legitimate configurations of Algorithm 1 can only be defined when the topology is stable as well as the direct confidences stabilized. Indeed, in case the direct ancestors or their direct confidences vary, no stabilization of the outputs can be obtained. Assuming these conditions are fulfilled, the set of legitimate configurations $\mathcal{L}_1$ of Algorithm 1 is defined by:

$$\forall c \in \mathcal{L}_1, \quad \forall v \in \mathcal{S}, \qquad \texttt{OUT}_v(c) = \textcircled{+}_{u \in \Gamma_v^{01}} \texttt{PRIV}_u(c)$$

Proposition 1. *Algorithm 1 is self-stabilizing: it converges in finite time to a legitimate configuration of $\mathcal{L}_1$ after the last occurrence of a transient fault and the last modification of either the topology or the direct confidences (inputs).*

Proof. Let e be an execution of Algorithm 1 on the distributed system $\mathcal{S}$. Let $c \in e$ be the first configuration from which the topology and the inputs are stable and such that there is no transient fault from c in e. Note that there is no more crash faults from c as they affect the topology. As there is no more transient faults from c, the private memories do contain the direct confidences. Let $c' \in e$ a configuration reachable from c such that, for any node v, its timer has expired

between c and c'. Then all the incoming memories have been purged (line 8). Let $c'' \in e$ a configuration reachable from c' such that, for any node v, its timer has expired between c' and c''. Since the topology is stable and there is no more transient fault, the direct confidence of each node u has been copied into the incoming memory of any node v such that u is a direct ancestor of v. Then any node v will compute $\textcircled{\cap}_{u \in \Gamma_v^{01}} \texttt{PRIV}_u(c)$ that will be stored in $\texttt{OUT}_v$. Hence, any configuration of e from c'' belongs to $\mathcal{L}_1$. □

3.5 Exploiting the output: from discretized weights to decision

By considering focal sets of cardinality larger than one (e.g., $\{\text{wet}, \text{cloud}\}$), the Theory of Belief Functions generalizes the Bayesian Probability Theory and is well adapted for representing weak states of knowledge. Nevertheless, when a decision has to be taken, one need to go back to focal sets of cardinality one. For this purpose the result BBA (expressed as a vector of weights) is converted in a mass function m and is then mapped to a *pignistic probability* function P [22], defined by: $P(A) = \sum_{\emptyset \neq B \subset \Omega}(B)\frac{|A \cap B|}{|B|}$.

Applying to our example, the decision would be: Is the umbrella necessary? By computing the pignistic probability on our previous numerical example, we find for instance that $P(\{sun\}) = 0.84$ when considering only the direct confidences of u and v while it is equal to 0.645 when considering the direct confidence of w, which did not agree with u and v.

4 Distributed Confidence Algorithm

Starting from Algorithm 1, we present an algorithm that computes on every node its *distributed confidence*, by combining the direct confidence of *all* the nodes, not only those of its neighbors.

4.1 Distributed Confidence Principle

The algorithm presented in the previous section is able to compute the so-called neighborhood confidence of every node by combining the direct confidence of its direct ancestors. The algorithm relies on local exchanges of belief functions represented as vector of masses belonging to $\mathbb{K}$. However, the information produced by a node will never impact nodes at more than one hop in the network. Yet in many cases it would be interesting to take into account remote information.

For instance, in our weather forecast example, the neighborhood confidence is preferable to the direct confidence because it relies on several measures. However it cannot determine the weather by advance. To the contrary, if remote measures are taken into account in the computation, a node could be warn about a depression before it arrives on it (supposing the distributed algorithm converges more rapidly than the wind!).

In order to preserve the networking and computing resources over the network, it is preferable that each node computes its distributed confidence using the one computed by its neighbors instead of using the direct confidence of remote nodes. By the way, the modification to be done in Algorithm 1 is at line 7: each node v will push its output $\texttt{OUT}_v$, containing the result of its local computation, instead of its input $\texttt{IN}_v$ containing its direct confidence.

7 **push(** $\texttt{OUT}_v$ **)**

This has some consequences on the distributed algorithm, and the operator has to be changed at line 5. We first introduce Algorithm 2a in Section 4.2, that uses the *cautious* operator. Then we show it cannot support transient failures and we introduce Algorithm 2b in Section 4.3, based on the cautious operator and a *discounting* function. This last one is self-stabilizing.

4.2 Cautious operator: Algorithm 2a

While it makes sense to use the Dempster operator to combine all the direct confidences, this is no more suitable with our algorithm modified at line 7 to push the output of each node. Indeed, whenever the network admits two distinct paths between nodes u and v, the direct confidence of u will be taken into account several times in the result built by node v. This problem is known as *data incest*. By the way, Algorithm 1 with the above mentioned modification at line 7 is only suitable for stable networks having a topology corresponding to a tree.

Besides the data incest, the algorithm would converge to the vector $\mathbf{w}_\perp \in \mathbb{K}$ (composed only with ϵ values) whenever there is a loop in the network because the multiplications by operator $*$ will converge to ϵ. As explained in [12], an idempotent operator is required for ensuring the convergence in a network with circuits.

In [6], an idempotent operator has been introduced for combining non dogmatic BBAs: the *cautious* operator denoted by $\oslash$. It is based on the *Least Commitment Principle*, which states that: "when several belief functions are compatible with a set of constraints, the least informative should be selected". When the BBAs are represented by weight functions, it is computed by taking the minimum of each component: $\mu_{1\oslash 2} = \mu_1 \oslash \mu_2$ is defined by $\mu_{1\oslash 2}(A) = \mu_1(A) \wedge \mu_2(A)$ for any $A \subsetneq \Omega$, where $\wedge$ denotes the minimum operator on $\mathbb{R}$.

Translated in $\mathbb{K}$, the discretization is here straightforward. We have, for any subset A of Ω with $A \neq \Omega$ and $A \neq \emptyset$, $\mathbf{w_1} \oslash \mathbf{w_2}[A] = \mathbf{w}_1[A] \wedge \mathbf{w}_2[A]$, where $\mathbf{w}[A]$ denotes the component of the vector $\mathbf{w}$ corresponding to the subset of A, and $\wedge$ the minimum operator on $\mathbb{W}$. This operator is associative, commutative and idempotent on $\mathbb{K}$. It admits $\mathbf{w}_\top$ as neutral element (vector composed only with some 1). It solves the data incest problem.

In fact, besides solving the data incest problem, operator $\oslash$ also ensures the termination of the distributed computation. Let Algorithm 2a be the algorithm obtained from Algorithm 1 with line 7 replaced by $\texttt{push(OUT}_v\texttt{)}$ and line 5 modified to use operator $\oslash$ instead of $\boxplus$. Let c_0 be the initial configuration defined

by: for all nodes v in $\mathcal{S}$, $\texttt{IN}_v$ is empty and $\texttt{OUT}_v = \mathbf{w}_\top$. Assuming the topology is stable and the direct confidences stabilized, the set of legitimate configurations $\mathcal{L}_2$ of Algorithm 2 is defined by:

$$\forall c \in \mathcal{L}_{2a}, \quad \forall v \in \mathcal{S}, \qquad \texttt{OUT}_v(c) = \textcircled{\&}_{u \in \Gamma_v} \texttt{PRIV}_u(c)$$

As the cautious operator is a law of an idempotent semi-group, the following proposition holds (Proposition 4 in [14]).

Proposition 2. *Algorithm 2a stabilizes in a fixed topology starting from configuration c_0, assuming the direct confidences (inputs) stabilizes.*

4.3 Cautious and discounting: Algorithm 2b

In contrast with Algorithm 1, Algorithm 2 stabilizes to a legitimate configuration only when it starts from the initial configuration c_0 (cf. Propositions 1 and 2). Indeed, an associative, commutative and idempotent operator leads to a self-stabilizing distributed algorithm only on networks corresponding to trees. For instance, consider a distributed system $\mathcal{S}$ in form of a loop composed of two nodes u and v and suppose that, due to a transient fault, the vector of weights $\mathbf{w}_\bot$ appears in the incoming memory of u. The next output of u will be $\mathbf{w}_\bot$, which will be sent to v. Both nodes will then converge to $\mathbf{w}_\bot$ whatever are their direct confidences (Proposition 7 in [14]).

On another hand, one may object that the legitimate configurations of Algorithm 2a are not always satisfactory. Indeed, it gives a single result per connected components of the network. When the information admits a local meaning (such as the weather forecast in our example), the result on a node u should differ from the result of a far node v except if all the nodes agree on their direct confidence. Hence, while it is useful to take into account remote information, all the nodes should not always converge to the same belief function.

We then introduce a discounting function $\mathbf{r}$, which is applied to each incoming data before the computation with the cautious operator (line 5). We call Algorithm 2b the algorithm obtained by modifications of the line 7 (for pushing $\texttt{OUT}_v$) and of the line 5 as follows:

5 $\quad \texttt{OUT}_v \leftarrow \texttt{OUT}_v \textcircled{\&} \mathbf{r}(\texttt{IN}_v[u])$

The function $\mathbf{r}$ is called a *discounting*; it is used to decrease the information in a given basic belief function. The choice of the discounting is application-dependent. Nevertheless, we impose two conditions on r. As $\textcircled{\&}$ is associative, commutative and idempotent, it defines an order relation denoted $\prec_{\textcircled{\&}}$ by: $\mathbf{w}_1 \prec_{\textcircled{\&}} \mathbf{w}_2$ if and only if $\mathbf{w}_1 \neq \mathbf{w}_2$ and $\mathbf{w}_1 \textcircled{\&} \mathbf{w}_2 = \mathbf{w}_1$.

Condition 1 *The discounting function $\mathbf{r}$ is an endomorphism of $(\mathbb{K}, \textcircled{\&})$: for any $\mathbf{w}_1$ and $\mathbf{w}_2$ in $\mathbb{K}$, $\mathbf{r}(\mathbf{w}_1)$ and $\mathbf{r}(\mathbf{w}_2)$ belong to $\mathbb{K}$ and $\mathbf{r}(\mathbf{w}_1 \textcircled{\&} \mathbf{w}_2) = \mathbf{r}(\mathbf{w}_1) \textcircled{\&} \mathbf{r}(\mathbf{w}_2)$.*

Condition 2 *The function $\mathbf{r}$ is expansive on $\mathbb{K}$: $\forall \mathbf{w} \in \mathbb{K} \setminus \{\mathbf{w}_\top\}$, $\mathbf{w} \prec_{\textcircled{\&}} \mathbf{r}(\mathbf{w})$ and $\mathbf{r}(\mathbf{w}_\top) = \mathbf{w}_\top$.*

Condition 1 is justified as follows. Consider a path $u_1, u_2, \ldots, u_k$ in a stable network and suppose that the algorithm has converged (all the outputs of the nodes do not change any more). Then we have: $\mathtt{OUT}_{u_k} = \mathtt{PRIV}_{u_k} \oslash r(\mathtt{OUT}_{u_{k-1}}) \oslash \cdots$. Recursively, $\mathtt{OUT}_{u_{k-1}} = \mathtt{PRIV}_{u_{k-1}} \oslash r(\mathtt{OUT}_{u_{k-2}}) \oslash \cdots$. Since r is an homomorphism, we have $\mathtt{OUT}_{u_k} = \mathtt{PRIV}_{u_k} \oslash r(\mathtt{PRIV}_{u_{k-1}}) \oslash r^2(\mathtt{PRIV}_{u_{k-2}}) \cdots$. Hence, thanks to Condition 1, the output of a node takes into account every received direct confidence a single time but discounted accordingly to the distance from the sender. The second condition is required for discounting the received BBA compared to the local direct confidence. It is also required for the convergence (else every node v in a loop would converge to $\mathbf{w}_\top$).

In our example, the weights being discretized up to the thousandth, the application $\mathbf{r} : \mathbf{w} \to \mathbf{r}(\mathbf{w})$ defined by $\mathbf{r}(\mathbf{w})[A] = \min(1, \mathbf{w}[A] + 0.1)$ for any $A \subset \Omega$ ($A \neq \Omega$ and $A \neq \emptyset$) is convenient.

4.4 Self-stabilizing property of Algorithm 2b

It is a remarkable result that the cautious operator along with a discounting is a strictly idempotent r-operator. Under certain conditions, the r-operators lead to self-stabilization of the global computation [12, 14, 13, 4]. This is a convenient way to design new self-stabilizing silent tasks: by only checking algebraic properties of the operator modeling the local computation, global properties over the whole networks are ensured.

An *r-operator* is the law of an r-semi-group [12], which generalizes the idempotent Abelian semi-group. Let $(\mathbb{S}, \diamond)$ be a set endowed by an operator $\diamond$ (*magma*). It admits a right-identity element $e_\diamond$ if $\forall x \in \mathbb{S}$, $x = x \diamond e_\diamond$. It is *weak left cancellative* iff $\forall y, z \in \mathbb{S}$, $(\forall x \in \mathbb{S}, x \diamond y = x \diamond z) \Leftrightarrow (y = z)$. Let $r : \mathbb{S} \to \mathbb{S}$ be a mapping. Then $(\mathbb{S}, \diamond)$ is *r-associative* iff $\forall x, y, z \in \mathbb{S}$, $x \diamond (y \diamond z) = x \diamond y \diamond r(z)$. It is *r-commutative* iff $\forall x, y \in \mathbb{S}$, $r(x) \diamond y = r(y) \diamond x$. It is *r-idempotent* iff $\forall x \in \mathbb{S}$, $r(x) \diamond x = r(x)$.

Definition 3 (r-semi-group). *Let $(\mathbb{S}, \triangleleft)$ be a weak left cancellative magma admitting the right identity element $e_\triangleleft$, and let $r : \mathbb{S} \to \mathbb{S}$ be an endomorphism. Then $(\mathbb{S}, \triangleleft)$ is an* r-semi-group *if it is r-associative, r-commutative, r-idempotent with the application r.*

Proposition 3. *Let $r : \mathbb{K} \to \mathbb{K}$ a mapping satisfying Conditions 1 and 2. Let $\triangleleft$ the operator defined on $\mathbb{K}$ by $\mathbf{w}_1 \circledcirc \mathbf{w_2} = \mathbf{w}_1 \oslash r(\mathbf{w}_2)$. Then $(\mathbb{K}, \circledcirc)$ is a strictly idempotent r-semi-group.*

Assuming the topology is stable and the direct confidences stabilized, the set of legitimate configurations $\mathcal{L}_{2b}$ of Algorithm 2b is defined by (we states $\mathrm{dist}(v, v) = 0$):

$$\forall c \in \mathcal{L}_{2b}, \quad \forall v \in \mathcal{S}, \qquad \mathtt{OUT}_v(c) = \circledcirc_{u \in \Gamma_v} \mathtt{PRIV}_u(c) = \oslash_{u \in \Gamma_v} \mathbf{r}^{\mathrm{dist(u,v)}}(\mathtt{PRIV}_u(c))$$

Proposition 4. *Algorithm 2b is self-stabilizing: it converges in finite time to a legitimate configuration of $\mathcal{L}_{2b}$ after the last occurrence of a transient fault and the last modification of either the topology or the direct confidences (inputs).*

Proof. As $\circledcirc$ induces a partial order relation on $\mathbb{K}$ ($\prec_{\circledcirc}$ based on $\wedge$ component per component of vectors), we apply results of [13], proved for the shared register model. As soon as the topology stabilizes, the distributed system $\mathcal{S}$ is assimilated to a shared-registers system, with the difference that links are unforeseen (not known at the beginning, not known by the senders and the receivers when the system is stabilized). Moreover, as the topology was moving, any value could have been copied in the incoming memories. This means that, during the stabilizing phase, Algorithm 2b runs on an unknown directed topology starting from *any* configuration. Thanks to Proposition 4 and Condition 2, Theorem 9 of [13] applies and Algorithm 2b is self-stabilizing. □

Let k be the integer defined by $r^k(\mathbf{w}_\perp) = \mathbf{w}_\top$ and D the diameter of the stabilized topology. Supposing a synchronous system, the stabilization time is $O(k + D)$. In a system without transient fault, when starting from the good initial configuration c_0 (§ 4.2), the convergence time is $O(D)$.

In our weather forecast example, our discounting function $\mathbf{r}$ satisfies $k = 10$.

5 Conclusion

In this paper, two algorithms have been presented for dealing with distributed imprecise an uncertain data in a network. The first one builds the neighborhood confidence of each node based on the inputs of its neighbors. The second one extends this computation to the whole network: each input is taken into account while favoring close information. These algorithms are self-stabilizing, meaning that, they converge in finite time in a legitimate configuration after the topology and the inputs become stable.

These results rely on the r-operators introduced for stabilizing distributed computations and on the cautious operator introduced for dealing with data incest in the Theory of Belief Functions, completed with a discounting for ensuring the self-stabilization and discretized for ensuring the convergence in finite time.

We believe that a large set of applications, either fundamental or practical, could take benefit of this approach. In particular, our simple weather forecast application is more efficient than other schemes based on data gathering while allowing to process uncertain and imprecise data given by cheap sensors. It supports crash faults of sensors, network reconfigurations and transient faults affecting memories. It can be implemented on wireless sensors networks.

Future work will concern extension of this approach as well as the study of its applications.

References

1. Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. Sharing memory robustly in message-passing systems. *Journal of the ACM*, 1(42):124–142, 1995.
2. V. Cherfaoui, T. Denoeux, and Z-L. Cherfi. *Vehicular Networks: Techniques, Standards, and Applications*, chapter Confidence management in Vehicular Network, pages 357–378. CRC Press, 2009. ISBN: 9781420085716.

3. V. Cherfaoui, T. Denoeux, and Z.L. Cherfi. Distributed data fusion: application to confidence management in vehicular networks. In *Proceedings of the 11^{th} International Conference on Information Fusion (FUSION 2008)*, Germany, 2008.
4. S. Delaët, B. Ducourthial, and S. Tixeuil. Self-stabilization with r-operators revisited. In *Journal of Aerospace Computing, Information, and Com.*, 2006.
5. A.P. Dempster. A generalization of bayesian inference. *Journal of the Royal Statistical Society*, 30:205–247, 1968.
6. T. Denœux. Conjunctive and disjunctive combination of belief functions induced by non distinct bodies of evidence. *Artificial Intelligence*, 172:234–264, 2008.
7. Y. Dieudonné, B Ducourthial, and S.-M. Senouci. Design and experimentation of a self-stabilizing data collection protocol for vehicular ad-hoc networks. IEEE Intelligent Vehicle Symposium 2012, Madrid, June 2012.
8. Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, 1974.
9. Shlomi Dolev. *Self-Stabilization.* MIT Press, 2000.
10. D. Dubois and H. Prade. Representation and combination of uncertainty with belief functions and possibility measures. *Computer intelligence*, 4:244–264, 1988.
11. Swan Dubois. *Tolerating Transient, Permanent, and Intermittent Failures.* PhD thesis, Université Pierre et Marie Curie, Paris, France, 2011.
12. B. Ducourthial. r-semi-groups: A generic approach for designing stabilizing silent tasks. In *SSS'2007*, Barcelona, November 2007.
13. B. Ducourthial and S. Tixeuil. Self-stabilization with path algebra. *Theor. Comput. Sci.*, 293(1):219–236, 2003.
14. Bertrand Ducourthial and Sébastien Tixeuil. Self-stabilization with r-operators. *Distributed Computing*, 14(3):147–162, 2001.
15. Andrea Gasparri, Flavio Fiorini, Maurizio Di Rocco, and Stefano Panzieri. A networked transferable belief model approach for distributed data aggregation. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, PP(99), 2011.
16. David L. Hall and James Llinas. *Handbook of Multisensor Data Fusion.* CRC Press, 2001.
17. A. Segall. Distributed network protocols. *IEEE Trans. Inf. Theory*, 29(1):23–34, 1983.
18. G. Shafer. *A mathematical theory of evidence.* Princeton, N.J, 1976.
19. Ph. Smets. The combination of evidence in the Ttransferable Belief Model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(5):447–458, 1990.
20. Ph. Smets. The canonical decomposition of a weighted belief. In *Int. Joint Conf. on Artificial Intelligence*, pages 1896–1901, San Mateo, Ca, 1995. Morgan Kaufman.
21. Ph. Smets. Data fusion in the transferable belief model. *Proceedings of. 3rd Intern. Conf. Information Fusion, Paris, France.*, 2000.
22. Ph. Smets. Decision making in the TBM: the necessity of the pignistic transformation. *Int. Journal of Approximate Reasoning*, 38:133–147, 2005.
23. Philippe Smets and Robert Kennes. The transferable belief model. *Artificial Intelligence*, 66:191–234, 1994.
24. G. Tel. *Topics in Distributed Algorithms*, volume 1 of *Cambridge International Series on Parallel Computation.* Cambridge University Press, 1991.
25. G. Tel. *Introduction to Distributed Algorithms.* Cambridge University Press, 1994.
26. N. El Zoghby, V. Cherfaoui, B. Ducourthial, and T. Denoeux. Distributed data fusion for detecting sybil attacks in VANETs. In *Proc. of the 2nd Int. Conference on Belief Functions*, Springer-Verlag, Advances in Intelligent and Software Computing, France, May 2012.