2008 American Control Conference
Westin Seattle Hotel, Seattle, Washington, USA
June 11-13, 2008

Las Vegas randomized algorithms in distributed consensus problems

Hideaki Ishii
Department of Computational Intelligence & Systems Science
Tokyo Institute of Technology, Yokohama 226-8502, Japan
Email: ishii@dis.titech.ac.jp

Roberto Tempo
IEIIT-CNR, Politecnico di Torino
Corso Duca degli Abruzzi 24, 10129 Torino, Italy
Email: roberto.tempo@polito.it

Abstract— **We consider distributed consensus problems from the viewpoint of probabilistic algorithms. In particular, we provide an overview on some specific problems where randomization is critical in achieving consensus among multiple agents. Further, we show that the randomized algorithms which are used in this setting are the so-called Las Vegas randomized algorithms (e.g. [24]). This class is different from that of Monte Carlo type, which has been recently successfully employed for various computationally difficult problems in systems and control. The objective of this paper is therefore to show the link between various distributed consensus problems and randomized algorithms for systems and control.**

I. Introduction

In recent years, distributed consensus, agreement, and flocking problems have gained much attention in the systems and control community. Control theoretic approaches have proven to be useful in the analysis of distributed systems for specifications such as stability and agreement. Recent references include [3], [4], [6], [7], [10], [12], [16], [18], [20], [21], [25], [27]. For additional details, we refer to [5] which gives a summary of the development of such problems along with some new results and to the special issue [1] which describes current research on this topic.

In particular, we consider the *average* consensus problems, which can be described as follows: There is a set of N agents that possess numerical values and communicate their values with their neighbors in an iterative way. The goal is that all agents eventually reach a common value, which is the average of the initial values of all agents. Such problems arise in applications related to multi-vehicle coordination, load balancing, and sensor networks.

The objective of this paper is to present the average consensus problems from the unifying viewpoint of probabilistic and Las Vegas algorithms. Recently, in the field of systems and control, techniques based on randomized algorithms have been developed (see, e.g, [23], [26]); however, we will see how the classes of algorithms there and those in the consensus problems perform differently. According to a formal definition used in computer science [17], a randomized algorithm is an algorithm that makes random choices during its execution to produce a result. This implies that even for the same input, the algorithm might produce different results at different runs, and moreover the results may even be incorrect.

More specifically, in this paper, we aim at clarifying two points: One is to introduce randomized algorithms appearing in several variations of average consensus problems. Such techniques are shown to be useful and can be crucial. Indeed, in some cases, there are even stronger results implying that no deterministic method can efficiently achieve consensus [9], [17]. The other is to show the difference in the classes of algorithms appearing in consensus problems and those in the probabilistic approach in control. To this end, we provide an overview on the probabilistic approach in control and see that the main algorithms there are the Monte Carlo type. Then, we show that the algorithms in the average consensus belong to the class of Las Vegas type algorithms. This class has recently been exploited for problems related to systems and control [11], [24].

The paper is organized as follows. In Section II, we present a general robustness analysis problem. In Section III, randomized algorithms of Monte Carlo type for such problems are discussed. In Section IV, we provide an introduction to Las Vegas algorithms. In Section V, we discuss three cases of average consensus problems and show the importance of probabilistic techniques. We conclude the paper in Section VI.

II. Probabilistic approach to uncertain systems

In the past decade, probabilistic methods for systems and control have significantly progressed. This research has been also motivated by results showing that many problems naturally arising in control are computationally difficult and are in fact NP-hard. Such problems can be found in areas including uncertain and hybrid systems. The development and application of probabilistic techniques to analysis and synthesis control problems have been proven effective yielding computationally efficient algorithms. For a detailed account on this topic, we refer to [23], [26].

In this section, we introduce a robustness analysis problem where various uncertainties can be represented. Given a system containing uncertain components, the objective of robustness analysis is to find whether certain control properties hold for all uncertainties. This general problem can be formulated as follows.

We first assume that the uncertainty in the system is represented by a real/complex matrix Δ and further that Δ belongs to a bounded set $\mathcal{B}$. On the other hand, the system property is measured by the *performance function* $J : \mathcal{B} \to \mathbb{R}$. The function J is assumed to be a measurable function. A general robustness analysis problem is to check whether a certain performance level γ is guaranteed for all possible uncertainties $\Delta \in \mathcal{B}$. In other words, we are

This work was supported in part by the Ministry of Education, Culture, Sports, Science and Technology, Japan, under Grant-in-Aid for Scientific Research No. 17760344.

interested in finding if

$$J(\Delta) \leq \gamma \text{ for all } \Delta \in \mathcal{B}. \tag{1}$$

If the uncertainty Δ is of general form, there are barriers in terms of computational complexity for solving this problem. We follow a probabilistic approach and shift the meaning of robustness from the deterministic sense as in (1) to a probabilistic one.

In the probabilistic approach, we assume that the uncertainty matrix $\boldsymbol{\Delta} \in \mathcal{B}$ is a random matrix; here, random variables are denoted in boldface letters. Let $\text{Prob}_{\boldsymbol{\Delta}}(\cdot)$ be the corresponding probability measure associated to Δ.

We consider two specific performance criteria using $J(\Delta)$. The first is the *worst-case performance* defined by

$$J_{\max} := \sup_{\Delta \in \mathcal{B}} J(\Delta). \tag{2}$$

The other is the *average-case performance*

$$J_{\text{ave}} := E_{\boldsymbol{\Delta}}(J(\boldsymbol{\Delta})),$$

where $E_{\boldsymbol{\Delta}}(J(\boldsymbol{\Delta}))$ denotes the expected value of the performance function with respect to the uncertainty set $\mathcal{B}$.

A performance function commonly employed in robustness analysis is the H^∞ norm of a closed-loop system. In this case, let the function $J(\Delta)$ be the norm of the system from the disturbance to the controlled output. Depending on the criterion of interest, we may choose to work with the worst-case performance or the average-case performance.

In this setting, a *decision problem* refers to a situation whose answer to an instance is either yes or no. Here, we address two uncertain decision problems: For a given performance level $\gamma > 0$, check if $J_{\max} \leq \gamma$ and $J_{\text{ave}} \leq \gamma$. We next show that these problems can be efficiently solved in a probabilistic sense.

III. Monte Carlo randomized algorithms

We now introduce Monte Carlo randomized algorithms. Most probabilistic results derived in systems and control are based on this type of algorithms. We will see later that this class is different from the one appearing in consensus problems. The definition is as follows [17].

Definition 1: A *Monte Carlo randomized algorithm (MCRA)* is a randomized algorithm that may produce a result that is incorrect (in the deterministic sense), but the probability of such an incorrect result is bounded.

In general, for an MCRA, the results and the running times would be different from one run to another since the algorithm is based on random sampling. As a consequence, the computational performance of such algorithms is usually measured by their *expected* running times. An MCRA is said to be *efficient* if the expected running time is of polynomial order in the problem size.

For decision problems, MCRAs can be divided into two classes based on how the error of the outputs are evaluated.

An MCRA for a decision problem is said to have *one-sided error* if it always provides a correct solution in one of the possible instances, but may provide a wrong solution for the other one [17].

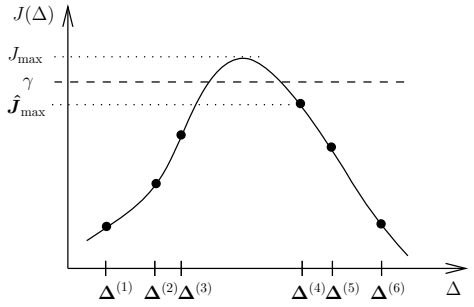

Fig. 1. One-sided MCRA: The worst-case performance when $\hat{\boldsymbol{J}}_{\max} < \gamma < J_{\max}$

The MCRA for the worst-case performance belongs to this class. This algorithm involves the computation of the *empirical maximum*, which is defined by

$$\hat{\boldsymbol{J}}_{\max} := \max_{i=1,2,\ldots,N} J(\boldsymbol{\Delta}^{(i)}),$$

where $\boldsymbol{\Delta}^{(i)} \in \mathcal{B}$, $i = 1, \ldots, N$, are independent and identically distributed (i.i.d.) samples of the random uncertainty matrix $\boldsymbol{\Delta}$ generated according to the probability measure $\text{Prob}_{\boldsymbol{\Delta}}$. Notice that the empirical maximum $\hat{\boldsymbol{J}}_{\max}$ is a random variable itself since its value depends on the random samples chosen for its computation. Fig. 1 shows a plot of $J(\Delta)$ and illustrates the concept for this problem.

This algorithm is a one-sided MCRA in the following sense. For a given performance level $\gamma > 0$, if $J_{\max} \leq \gamma$, then clearly the probability that the algorithm outputs the correct answer is one. Hence, this algorithm always provides a correct solution for this instance. On the other hand, if $J_{\max} > \gamma$, then the probability of obtaining $\hat{\boldsymbol{J}}_{\max} \leq \gamma$ is nonzero. This implies that, for this instance, the algorithm can give an erroneous result with a nonzero probability.

Now, noticing that the empirical maximum $\hat{\boldsymbol{J}}_{\max}$ is always smaller than $J_{\max}$, we shall pose a natural question, how well does $\hat{\boldsymbol{J}}_{\max}$ estimate the true maximum? Under a sufficiently large sample size N, a probabilistic statement can be made; see, e.g., [22], [23].

Another class of MCRAs for decision problems is that of *two-sided error* algorithms. Such algorithms may produce a wrong solution for both instances when the answer is yes and no [17]. The average-case problem is an example of a two-sided MCRA; for details, we refer to [24].

IV. Las Vegas randomized algorithms

We introduce the Las Vegas algorithms and its basic properties. This type has not been employed much in the context of control, but is important for consensus problems.

A. Preliminaries

The formal definition is given in [17] as follows:

Definition 2: Las Vegas randomized algorithms (LVRA) are randomized algorithms which always give the correct

answer. The only difference from one run to another is the running time.

For obvious reasons, such algorithms are also called *zero-sided error* Monte Carlo. Because of randomization, the running time is random (similarly to MCRA) and may be different in each execution. Hence, the expected running time is of interest. We note that the expectation is with respect to the random samples generated during each run and not to the input of the algorithm. Furthermore, if the expected running time is of polynomial order in the problem size, the algorithm is said to be *efficient*.

A well-known example is the *Randomized Quick Sort (RQS)* [14], [17] described in the following.

Example 1: Given a set $\mathcal{S}_1 = \{x_1, \ldots, x_N\}$ of N real numbers, consider the problem of sorting the numbers in an increasing order. The RQS is a randomized algorithm that solves this problem in a computationally efficient way. The outline of the algorithm is as follows

1) Randomly select a number $\boldsymbol{x}^{(1)}$ in the set $\mathcal{S}_1$.
2) Perform deterministic comparisons between $\boldsymbol{x}^{(1)}$ and other elements in $\mathcal{S}_1$. Let $\boldsymbol{\mathcal{S}}^{(2)}$ be the set of numbers smaller than $\boldsymbol{x}^{(1)}$, and let $\boldsymbol{\mathcal{S}}^{(3)}$ be the set of numbers larger than $\boldsymbol{x}^{(1)}$.
3) Recursively apply the two steps above to the sets $\boldsymbol{\mathcal{S}}^{(2)}$ and $\boldsymbol{\mathcal{S}}^{(3)}$. Output the sorted version of $\boldsymbol{\mathcal{S}}^{(2)}$, $\boldsymbol{x}^{(1)}$, and then the sorted version of $\boldsymbol{\mathcal{S}}^{(3)}$.

The rationale for randomization in 1) is as follows: The algorithm would be most efficient if the original set $\mathcal{S}_1$ is divided into two sets having the same cardinalities. However, this requires the median of $\mathcal{S}_1$ as $\boldsymbol{x}^{(1)}$, which is costly to find. To randomly choose $\boldsymbol{x}^{(1)}$ is a simple strategy, but on average provides a good estimate of the median.

In a formal analysis, the running time is measured by the number of comparisons. It follows that the expected running time is of order $O(N \log N)$; in fact, the running time is of this order with high probability, at least $1 - 1/N$ [15]. The RQS is more efficient than, for example, a deterministic brute-force approach, which has complexity $O(N^2)$.

For the RQS, the worst case is when the randomly chosen number in 1) always happens to be either the smallest or the largest in the set. Then, the running time achieves the order $O(N^2)$. The RQS is, however, recognized as one of the most useful general purpose sorting algorithms [14]. $\triangledown$

As mentioned above, Las Vegas algorithms are randomized algorithms that always produce correct results. It is clear that this feature can be expected for only a limited number of Monte Carlo type algorithms. Hence, their application is naturally limited. One such algorithm is developed for a switched systems problem in [11].

B. LVRAs for uncertain decision problems

We now present a discussion parallel to that on uncertain systems in Sections II and III for Las Vegas algorithms.

First, as the uncertainty set, we take a finite subset $\tilde{\mathcal{B}}$ of $\mathcal{B}$ with N elements given as

$$\tilde{\mathcal{B}} = \left\{\tilde{\Delta}_1,\ \tilde{\Delta}_2, \ldots,\ \tilde{\Delta}_N\right\} \subset \mathcal{B}.$$

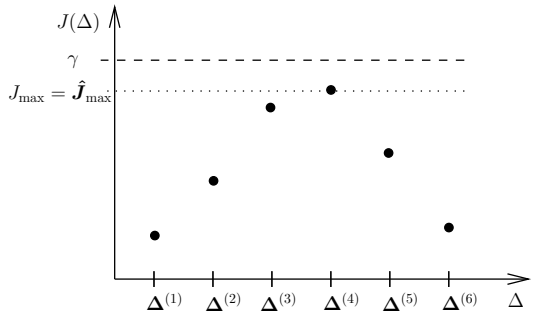

Fig. 2. LVRA: The worst-case performance when $J_{\max} = \hat{\boldsymbol{J}}_{\max} < \gamma$

Assuming that the uncertain matrices in $\tilde{\mathcal{B}}$ are random variables, we consider a discrete probability measure; as in Section II, this measure is denoted by $\mathrm{Prob}_{\boldsymbol{\Delta}}$. Similarly, let the performance function be $J : \tilde{\mathcal{B}} \to \mathbb{R}$. The general robustness analysis problem is to find whether, for a given performance level $\gamma > 0$, $J(\Delta) \leq \gamma$ for all $\Delta \in \tilde{\mathcal{B}}$. The corresponding worst-case performance is

$$J_{\max} := \max_{\Delta \in \tilde{\mathcal{B}}} J(\Delta), \tag{3}$$

and the average case performance is

$$J_{\text{ave}} := E_{\boldsymbol{\Delta}}[J(\boldsymbol{\Delta})] = \sum_{i=1}^{N} J(\tilde{\Delta}_i)\mathrm{Prob}_{\boldsymbol{\Delta}}(\tilde{\Delta}_i).$$

We now consider the uncertain decision problem for the worst-case performance. Given a scalar $\gamma > 0$, check whether $J_{\max} \leq \gamma$. The LVRA for this case is as follows: Let $\tilde{\mathcal{B}}_0 = \tilde{\mathcal{B}}$ and let $k = 1$. Also, let $\bar{J}^{(0)} = -\infty$. At the kth step, randomly select a sample $\boldsymbol{\Delta}^{(k)}$ from $\tilde{\mathcal{B}}_{k-1}$. Then, set $\bar{\boldsymbol{J}}^{(k)} = \max\{J(\boldsymbol{\Delta}^{(k)}),\ \bar{\boldsymbol{J}}^{(k-1)}\}$ and $\tilde{\mathcal{B}}_k = \tilde{\mathcal{B}}_{k-1} \setminus \{\boldsymbol{\Delta}^{(k)}\}$, and go to the next step. After the Nth step, the maximum performance over the samples is given by $\hat{\boldsymbol{J}}_{\max} = \bar{\boldsymbol{J}}^{(N)}$.

This algorithm always outputs the correct answer and hence is a Las Vegas type. For either problem instance ($J_{\max} \leq \gamma$ or $J_{\max} > \gamma$), the value $\hat{\boldsymbol{J}}_{\max}$ that the algorithm produces coincides with the true value $J_{\max}$. This is illustrated in Fig. 2.

When N is large, the computational complexity can be relaxed by modifying the algorithm just described. The resulting algorithm is a Monte Carlo type. This can be done by stopping at the kth step with $k < N$ and by computing the maximum performance over the k samples. In fact, the resulting algorithm becomes a one-sided MCRA. We remark that this approach is closely related to the ordinal optimization; see, e.g., [8]. The objective there is to find not the maximum performance value but the value that is at least within the mth largest.

The average-case performance can be similarly discussed.

V. Las Vegas randomized algorithms for distributed average consensus

In this section, we present several problems in distributed average consensus where the application of Las Vegas type algorithms can be effective and sometimes in fact crucial.

A. General problem setup

Consider a network of N nodes specified by the graph $(\mathcal{V}, \mathcal{E})$, where $\mathcal{V} := \{1, 2, \ldots, N\}$ is the set of nodes and $\mathcal{E}$ is the set of edges. The graph is assumed to be undirected and connected (see, e.g., [17] for an introduction to random graph theory); this means that the edges are not associated with directions and that for any $i, j \in \mathcal{V}$, there is a path that connects the nodes i and j. At time k, each node i has a scalar value $x_i(k)$ whose initial value is $x_i(0)$.

The goal is to provide an algorithm such that (i) the nodes update their values $x_i(k)$ using the information communicated from their neighbors and (ii) the values of the nodes eventually converge to the average of the initial values. To this end, for a consensus algorithm, there are two elements that need to be determined: The rules for the nodes to update their values and the neighbors with which each node should communicate.

This problem is a particular version of distributed consensus. In general, consensus problems do not require to which number the values of the nodes must converge, but only that the number should be the same for all nodes. In the following, we consider three cases of this problem. The difference is in the range of the node values: Real numbers, integer (quantized) numbers, and binary numbers.

We introduce some notation that will be used throughout this section. Let the N-dimensional vector consisting of all node values at time k be $x(k) = [x_1(k) \cdots x_N(k)]^T$. The communication pattern for the nodes at time k is specified by the edge set $\tilde{\mathcal{E}}(k) \subset \mathcal{E}$, i.e., if $\{i, j\} \in \tilde{\mathcal{E}}(k)$, then $x_i(k)$ is updated using $x_j(k)$ and vice versa; in this case, the nodes i and j are *neighbors* of each other at this time. In general, neighbors of a node may change over the time.

B. Real-valued case

We first present the case when the node values take real numbers which is studied in [28]. We say that *average consensus* (in a deterministic sense) is achieved if the following condition is satisfied:

$$\lim_{k\to\infty} x_i(k) = \frac{1}{N}\sum_{j=1}^{N} x_j(0) \quad \text{for all } i = 1, \ldots, N. \tag{4}$$

The update rule for the node i takes a linear form as

$$x_i(k+1) = W_{ii}(k)x_i(k) + \sum_{j\in\mathcal{N}_i(k)} W_{ij}(k)x_j(k), \tag{5}$$

where $\mathcal{N}_i(k) := \{j : \{i, j\} \in \tilde{\mathcal{E}}(k)\}$ is the set of neighbors for node i at time k, and $W_{ij}(k)$ are the weights. The weights are time varying and are specified by

$$W_{ij}(k) = \begin{cases} \frac{1}{1+\max\{d_i(k),\, d_j(k)\}} & \text{if } \{i, j\} \in \tilde{\mathcal{E}}(k), \\ 1 - \sum_{l\in\mathcal{N}_i(k)} W_{il}(k) & \text{if } i = j, \\ 0 & \text{otherwise,} \end{cases} \tag{6}$$

where $d_i(k)$ denotes the cardinality of $\mathcal{N}_i(k)$, that is, the number of neighbors for the node i. Notice that the sum of the weights at each node is one; this makes the matrix W a stochastic one. In [28], it is remarked that the weights $W_{ij}(k)$ in (6) are adopted from the Metropolis algorithm used in Markov chain Monte Carlo.

The update rule in (5) can be implemented in a distributed and causal manner. This is because the nodes only require the information of the values coming from their neighbors at the time. Regarding the communication pattern specified by $\tilde{\mathcal{E}}(k)$, $k \in \mathbb{Z}_+$, the assumption is as follows: The graph $(\mathcal{V}, \cup_{s\geq k}\tilde{\mathcal{E}}(s))$ is a connected graph for all k. In words, this says that the collection of edge sets occurring infinitely often over the time makes it a connected graph.

The following is the main result of [28].

Theorem 1: Under the update rule in (5) and the communication pattern satisfying the condition that the graph $(\mathcal{V}, \cup_{s\geq k}\tilde{\mathcal{E}}(s))$ is a connected graph for all k, distributed average consensus in the sense of (4) is achieved for each initial condition $x(0) \in \mathbb{R}^N$.

This theorem provides a condition on $\tilde{\mathcal{E}}(k)$ which must be specified at the time of implementation for the average consensus in a deterministic sense. Similar conditions are employed in other schemes in, e.g., [5], [12], [16].

A simple way to implement a communication pattern with the desired property is to employ randomization: Each node i communicates with a randomly and independently chosen neighbor $\boldsymbol{j}_k$ satisfying $\{i, \boldsymbol{j}_k\} \in \mathcal{E}$ at time k. In particular, we allocate positive probability to each edge $\{i, j\}$ in $\mathcal{E}$. Since $(\mathcal{V}, \mathcal{E})$ is a connected graph, the condition on the communication pattern holds probabilistically. That is, for each k, the probability that the graph $(\mathcal{V}, \cup_{s\geq k}\tilde{\boldsymbol{\mathcal{E}}}(s))$ is connected is one. Hence, the resulting algorithm achieves average consensus in (4) with probability one for any $x(0)$.

A stochastic average consensus scheme is also proposed in [10], where the graph edges are selected randomly and independent of each other. The paper provides a probabilistic analysis of the convergence. The scheme is based on a sampled-data communication protocol and employs weights that are different from those in (6). Other works exploiting randomized communication patterns include [7], [21], [27].

C. Quantized-valued case

We next look at the average consensus for the quantized-valued case. This scheme is proposed in [13]. Here, by quantized, we mean that the node values are integers. This consensus problem requires a somewhat different treatment from the real-valued case. To begin with, the average of the initial values may not be an integer. Thus, the target value is an integer approximation of the true average and is not necessarily unique. Moreover, consensus can be achieved in finite time because the nodes are updated in integers at each time instant.

More specifically, the algorithm is said to achieve *quantized average consensus* if the following conditions hold:

(i) The values are integers at all times: $x_i(k) \in \mathbb{Z}$, $\forall i, k$.

(ii) The sum of the node values remains constant: $\sum_{i=1}^{N} x_i(k) = \sum_{i=1}^{N} x_i(0)$ for all k.

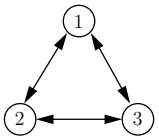

Fig. 3. A graph of 3 nodes

(iii) All values converge to the quantized average: There exists k^* such that $x_i(k) \in \{\bar{x}, \bar{x}+1\}$ for all $k > k^*$ and i, where $\bar{x} = \left\lfloor \sum_{i=1}^{N} x_i(0)/N \right\rfloor$.

In [13], a class of randomized algorithms called *quantized gossip algorithms* is proposed; see also [6]. The following are the requirements for such algorithms: At time k, one edge $\{\boldsymbol{i}, \boldsymbol{j}\} \in \mathcal{E}$ is selected at random in an i.i.d. fashion. Let $\boldsymbol{D_{ij}}(k) = |\boldsymbol{x_i}(k) - \boldsymbol{x_j}(k)|$.

(a) If $\boldsymbol{D_{ij}}(k) = 0$, then the values of the nodes $\boldsymbol{i}$ and $\boldsymbol{j}$ remain the same for time $k+1$.

(b) If $\boldsymbol{D_{ij}}(k) = 1$, then the values are exchanged, or *swapped*, by

$$\boldsymbol{x_i}(k+1) = \boldsymbol{x_j}(k) \quad \text{and} \quad \boldsymbol{x_j}(k+1) = \boldsymbol{x_i}(k). \quad (7)$$

(c) Otherwise, the updates in the values satisfy

$$\boldsymbol{x_i}(k+1) + \boldsymbol{x_j}(k+1) = \boldsymbol{x_i}(k) + \boldsymbol{x_j}(k),$$
$$\boldsymbol{D_{ij}}(k+1) < \boldsymbol{D_{ij}}(k).$$

Notice that algorithms in this class are by definition randomized. One way of specifying the distribution for choosing the edges is to allocate positive probabilities to all edges in the graph. We refer the readers to [13] for several explicit quantized gossip algorithms that ensure the requirements above. We also emphasize that these algorithms are different from the one for the real-valued case in Section V-B. Especially, the swapping in (7) becomes crucial when the node values are close to the average.

The next result applies to any such algorithm [13].

Theorem 2: For each initial condition $x(0) \in \mathbb{Z}^N$, a quantized gossip algorithm achieves quantized average consensus in a finite number of steps with probability one.

An interesting aspect of this algorithm is that randomization is essential. This can be illustrated through an example given in [13]. Consider the graph with three nodes in Fig. 3. Initially, the node i is given the value i for $i = 1, 2, 3$ and, thus, the average value is 2. Suppose that we employ a deterministic, periodic scheme for the edge selection with period 3 by following $\{1, 2\}$, $\{1, 3\}$, $\{3, 2\}$, $\{1, 2\}, \ldots$. Under this scheme, it is clear that only swapping in (7) will take place for all times. Hence, the set of node values remains 1, 2, and 3 and will never reach the consensus values. In contrast, by randomly choosing the edges, average consensus is possible in a matter of a few steps. This is an attractive communication scheme for its simplicity.

We remark that in [13] further probabilistic analysis is given on the expected running time for achieving quantized consensus. In particular, if the graph is either fully connected or linear (i.e., the nodes are connected as $1 - 2 - \cdots - N$), then the expected running time is polynomial with respect to the number of nodes. Hence, this algorithm is an example of an efficient Las Vegas algorithm. This is in contrast with the real valued case in Section V-B, where in general convergence of the node values to the average is guaranteed only asymptotically.

D. Binary-valued case

The third consensus problem we study is when the agents take binary values $x_i(k) \in \{0, 1\}$ for all i, k. In particular, the problem we discuss here is known as the *Byzantine agreement* problem in the field of distributed computing, see e.g. [17]. We will show that in this case even stronger results in favor of randomized schemes can be obtained.

A distinct feature of this problem is that some of the N agents are faulty. Such agents may try to deceive the nonfaulty agents and may even communicate to each other secretly. We assume that there are N_f such agents, which are fixed but their identity is not known to the nonfaulty ones. Regarding the communication pattern, the graph is fully connected. Each agent sends a message to all others at each time k; faulty agents may send different messages to others. Randomization hence takes part not in the communication pattern. Instead, a *global coin toss* is performed by a trusted party who sends the (true) result to all agents at each k.

The objective is to achieve a form of average consensus even in the worst case. We say that *binary consensus* is achieved if each agent i determines the *decision value* $y_i \in \{0, 1\}$ such that

(i) all nonfaulty agents arrive at the same decision value;
(ii) if all nonfaulty agents have the same initial value $x_i(0)$, then they finish with $y_i = x_i(0)$.

We consider the simple case where N is a multiple of 8 and $N_f < N/8$. We also let

$$L := 5N/8 + 1, H := 3N/4 + 1, G := 7N/8.$$

The algorithm for each agent i we now present is due to [19]; basically, the agent sends its value and decides on the majority of the values that it receives.

1) At time k, send the value $\boldsymbol{x}_i(k)$ to other agents and receive $\boldsymbol{x}_j(k)$, $j \neq i$, from them.
2) Set the *majority value* $\boldsymbol{m}_i(k) \in \{0, 1\}$ to what the majority of agents sent as their values. Then, set the *tally* $\boldsymbol{t}_i(k)$ equal to the number of agents whose values are the same as $\boldsymbol{m}_i(k)$.
3) Now, depending on the result of the coin toss, let the *threshold* $\bar{\boldsymbol{t}}(k)$ be L if the coin shows heads and H otherwise (note that the threshold is the same for all agents).
4) Set the value $\boldsymbol{x}_i(k)$ to the majority value $\boldsymbol{m}_i(k)$ if $\boldsymbol{t}_i(k) \geq \bar{\boldsymbol{t}}(k)$ and to 0 otherwise.
5) If the tally satisfies $\boldsymbol{t}_i(k) \geq G$, then let the decision value $\boldsymbol{y}_i$ be equal to $\boldsymbol{m}_i(k)$.

There are two simple situations. (a) When all nonfaulty agents have the same value at one point, all of them will

finish with this value as their decision values in a constant number of steps. (b) When two nonfaulty agents have different majority values; then, all values $\boldsymbol{x}_i(k)$ will become 0 during this step because none of the tallies exceeds the threshold (in step 4 above).

The interesting case is when all nonfaulty agents have the same majority value. Then, the faulty agents have a chance to confuse them. This can be done by making some nonfaulty agents have tallies exceeding the threshold and others have tallies smaller than the threshold. However, under the scheme, the chance is limited. Since $H-L \geq N_f$, the faulty agents can deceive any agent only for one of the threshold values, L or H. Further, the threshold randomly varies over L and H because of the coin toss. Consequently, the probability of deceiving is 1/2. On the other hand, when they fail to deceive, all nonfaulty agents take the same value, which leads to consensus.

The main result concerning the Byzantine agreement problem is given below [17].

Theorem 3: For the algorithm presented above, binary consensus is achieved with probability one for each initial condition $x(0) \in \{0,1\}^N$. Moreover, the expected number of steps required is a constant.

This theorem implies that the consensus algorithm is an efficient Las Vegas type. In contrast, it is known that any deterministic algorithm requires $N_f + 1$ steps [17]. It is emphasized that even stronger results have been obtained in the area of distributed computing. One is the so-called *impossible result* for an asynchronous version of the binary consensus problem [9]. This result states that if there is no synchronized clock owned by the agents and if there is no assumption on the sampling periods of the agents (i.e., the processing speed may be different), for any deterministic algorithm, it is impossible to achieve consensus. Randomized schemes such as using a global coin toss have been shown to be effective in this case as well. For a survey on this topic, which also includes many recent improvements on the original algorithm, we refer to [2].

VI. Conclusion

In this paper, we discussed several variations of the average consensus problem and the role of probabilistic algorithms in this context. The class of Las Vegas randomized algorithms, which has been recently employed in the field of systems and control [11], [24], has been shown to be effective and sometimes crucial. Future research will deal with other consensus problems where the agents take real/quantized values and may be faulty.

Acknowledgment: The authors would like to thank Teodoro Alamo, Tamer Başar, Ivan Cibrario Bertolotti, Yasumasa Fujisaki, Akshay Kashyap, and Sandro Zampieri for the interesting discussions and for the valuable comments.

References

[1] P. J. Antsaklis and J. Baillieul, Guest Editors. Special Issue on the Technology of Networked Control Systems. *Proc. IEEE*, 95(1), 2007.

[2] J. Aspnes. Randomized protocols for asynchronous consensus. *Distributed Computing*, 16:165–175, 2003.

[3] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, Englewood Cliffs, NJ, 1989.

[4] D. P. Bertsekas and J. N. Tsitsiklis. Comments on "Coordination of groups of mobile autonomous agents using nearest neighbor rules". *IEEE Trans. Autom. Control*, 52:968–969, 2007.

[5] V. D. Blondel, J. M. Hendrickx, A. Olshevsky, and J. N. Tsitsiklis. Convergence in multiagent coordination, consensus, and flocking. In *Proc. 44th IEEE Conf. on Decision and Control and European Control Conf.*, pages 2996–3000, 2005.

[6] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Randomized gossip algorithms. *IEEE Trans. Information Theory*, 52:2508–2530, 2006.

[7] R. Carli, F. Fagnani, M. Focoso, A. Speranzon, and S. Zampieri. Communication constraints in the average consensus problem. *Automatica*, 44:671–684, 2008.

[8] M. Deng and Y.-C. Ho. An ordinal optimization approach to optimal control problems. *Automatica*, 35:331–338, 1999.

[9] M. J. Fisher, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty processor. *J. ACM*, 32:374–382, 1985.

[10] Y. Hatano and M. Mesbahi. Agreement over random networks. *IEEE Trans. Autom. Control*, 50:1867–72, 2005.

[11] H. Ishii and R. Tempo. Probabilistic sorting and stabilization of switched systems. Submitted for publication, 2007.

[12] A. Jadbabaie, J. Lin, and A. S. Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Trans. Autom. Control*, 48:988–1001, 2003.

[13] A. Kashyap, T. Başar, and R. Srikant. Quantized consensus. *Automatica*, 43:1192–1203, 2007.

[14] D. E. Knuth. *The Art of Computer Programming,* 2nd edition, volume 3: Sorting and Searching. Addison-Wesley, Reading, MA, 1998.

[15] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[16] L. Moreau. Stability of multiagent systems with time-dependent communication links. *IEEE Trans. Autom. Control*, 50:169–182, 2005.

[17] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[18] R. Olfati-Saber and R. Murray. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Trans. Autom. Control*, 49:1520–1533, 2004.

[19] M. O. Rabin. Randomized Byzantin generals. In *Proc. Annual Symp. on Foundations of Computer Science*, pages 403–409, 1983.

[20] A. V. Savkin. Coordinated collective motion of groups of autonomous robots: Analysis of Vicsek's model. *IEEE Trans. Autom. Control*, 49:981–983, 2004.

[21] A. Tahbaz-Salehi and A. Jadbabaie. Necessary and sufficient conditions for consensus over random independent and identically distributed switching graphs. In *Proc. 46th IEEE Conf. on Decision and Control*, pages 4209–4214, 2007.

[22] R. Tempo, E. W. Bai, and F. Dabbene. Probabilistic robustness analysis: Explicit bounds for the minimum number of samples. *Systems & Control Letters*, 30:237–242, 1997.

[23] R. Tempo, G. Calafiore, and F. Dabbene. *Randomized Algorithms for Analysis and Control of Uncertain Systems*. Springer, London, 2005.

[24] R. Tempo and H. Ishii. Monte Carlo and Las Vegas randomized algorithms for systems and control: An introduction. *European J. Control*, 13:189–203, 2007.

[25] J. N. Tsitsiklis. *Problems in Decentralized Decision Making and Computation*. PhD thesis, Dept. of Electrical Engineering and Computer Science, MIT, 1984. http://web.mit.edu/jnt/www/Papers/PhD-84-jnt.pdf.

[26] M. Vidyasagar. Statistical learning theory and randomized algorithms for control. *IEEE Control Systems Magazine*, 18(6):69–85, 1998.

[27] C. W. Wu. Synchronization and convergence of linear dynamics in random directed networks. *IEEE Trans. Autom. Control*, 51:1207–1210, 2006.

[28] L. Xiao, S. Boyd, and S. Lall. A scheme for robust distributed sensor fusion based on average consensus. In *Proc. Conf. on Information Processing in Sensor Networks*, pages 63–70, 2005.