IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Distributed Interval Synchronization

ANDREA FUSIELLO, (Senior Member, IEEE), and PIER LUCA MONTESSORO(Member, IEEE)
DPIA, University of Udine Via delle Scienze 206, 33100 Udine, Italy
Corresponding author: Andrea Fusiello (e-mail: name.surname@uniud.it).

ABSTRACT In this paper we address the problem of using pairwise measures associated with the edges of a graph to obtain absolute consistent measures associated with the nodes. This problem is known in the literature as graph synchronisation. In particular, we rigorously deal with the uncertainty affecting the measures thanks to the interval analysis approach. We propose an asynchronous, distributed algorithm that converges to an interval solution that represents all possible sharp solutions consistent with the measures.

INDEX TERMS Uncertainty in measurements, Synchronization, Interval Analysis, Distributed algorithms

I. INTRODUCTION

THIS paper deals with the synchronisation problem, which can be stated in its most general form as follows: given a network of nodes, where each node is characterised by an unknown label and pairs of nodes can measure the difference between their labels, the goal is to estimate the unknown labels from the pairwise measurements. The problem can be modelled as a graph in which the unknown labels are assigned to the nodes and the edges encode the pairwise measurements, and it is well-posed only if such a graph is connected. The solution is obtained from a linear system whose coefficient matrix is the incidence matrix of the graph.

The term synchronisation [9] originally refers to the case where the labels are real numbers representing local times and the goal is to align, synchronise, all these clocks to a common time, starting from local measures of time differences. Topographic levelling [2] is also in fact a synchronisation, and – more generally – the labels are not restricted to real numbers but can take values in any group (see, for example, [3]).

Although our method is general, in this paper we will refer to the problem of clock synchronisation in sensor networks in order to anchor it in a real application.

Synchronisation is a common problem in distributed systems (e.g. wireless sensor networks) [19], [22], [23]. In most cases, there is no global knowledge of the graph topology, edge labels and node labels [10], [17], [20], so we adopt a distributed method inspired by multi-agent systems: each node evaluates its label asynchronously only on the basis of its neighbourhood. Asynchronous distributed algorithms are robust against packet loss, node failures, and many other problems that occur in large wireless sensor networks [8].

The measure of the differences attached to the edges of the graph is affected by uncertainty (see e.g. [1] for the case of clock synchronisation), which we model using intervals, assuming that the probability that the value of the measure lies within the interval for all practical purposes is equal to one, and the probability that it lies outside the interval is essentially zero. There is no specific assumption about the probability distribution of the measure within the interval. We propagate uncertainty using the rules of Interval Analysis (IA), as in, e.g., [16], [24] that guarantees that the resulting intervals contain the true values, if the initial assumption is verified. IA has been used in several engineering applications, such as Computer Vision [6], [7] and Robotics [12], [13], [21], to mention some.

The baseline solution would be the straightforward application of IA rules to the synchronisation over real numbers. Our algorithm iteratively converges to an interval solution that encloses the true one and is tighter than the baseline method, as confirmed by simulations. To the best of our knowledge, this is the first synchronisation algorithm working with intervals.

II. BACKGROUND

In this section we will briefly review some background notions of interval analysis (Sec. II-A) and synchronisation (Sec. II-B).

A. INTERVAL ANALYSIS

Interval analysis (IA) [14], [15] is an approach to solving numerical problems that involves performing calculations on sets of real numbers rather than on floating-point approximations of them. IA defines methods for calculating an interval enclosing the range of various elementary mathematical functions. It was introduced to limit measurement errors of physical quantities for which no statistical distribution was known. Another important application of IA is the construc-

tion of verifiable solutions to constraints that return intervals guaranteed to contain all real solutions.

Interval enclosures are provably super-sets of the mathematically correct results, which is why the interval approach is said to be rigorous. Adhering to the IA paradigm, one should not consider any probability distributions inside the intervals.

There are two main advantages of IA over classical numerical analysis. The first is that input errors and rounding errors are automatically included in the interval result. Thus interval estimation can be viewed as automatically performing both computation and error analysis. The second is that IA allows to compute upper and lower bounds on the range of a function over an interval, and this proves useful in solving global optimisation problems.

In the sequel of this section we shall follow the notation used in [11], where intervals are denoted by boldface. Underscores and overscores will represent respectively lower and upper bounds of intervals. The midpoint of an interval $\mathbf{x}$ is denoted by $\mathrm{mid}(\mathbf{x})$ and its radius (equal to half of the interval width) by $\mathrm{rad}(\mathbf{x})$. $\mathbb{IR}$ and $\mathbb{IR}^n$ stand respectively for the set of real intervals and the set of real interval vectors of dimension n.

If $\mathbf{x} = [\underline{x}, \overline{x}]$ and $\mathbf{y} = [\underline{y}, \overline{y}]$, a binary operation between $\mathbf{x}$ and $\mathbf{y}$ is defined in interval arithmetic as:

$$\mathbf{x} \circ \mathbf{y} = \{x \circ y \mid x \in \mathbf{x} \wedge y \in \mathbf{y}\}, \forall \circ \in \{+, -, \times, \div\}.$$

Operationally, interval operations are defined by the min-max formula:

$$\mathbf{x} \circ \mathbf{y} = \big[\min\{\underline{x} \circ \underline{y}, \underline{x} \circ \overline{y}, \overline{x} \circ \underline{y}, \overline{x} \circ \overline{y}\}, \max\{\underline{x} \circ \underline{y}, \underline{x} \circ \overline{y}, \overline{x} \circ \underline{y}, \overline{x} \circ \overline{y}\}\big]. \quad (1)$$

Here, interval division $\mathbf{x}/\mathbf{y}$ is undefined when $0 \in \mathbf{y}$.

It should be noted that the ranges of the four elementary interval operations are exactly the same as the ranges of the corresponding real operations, provided that the endpoints are rounded outward. In general, for arbitrary functions, interval calculations do not return the exact range, but only an overestimation. Furthermore, different expressions for the same function give different results, although all are guaranteed to contain the exact range.

B. SYNCHRONISATION IN $(\mathbb{R}, +)$

In this section we will give a brief account of the synchronisation in the group of real numbers $\mathbb{R}$ with the sum (see also [18]).

Let $G = (V, E)$ a be a directed simple graph, whose nodes are $V = \{v_1 \dots v_n\}$ and edges are $E \subseteq V \times V$ with $|E| = m$. A node labelling $x : V \to \mathbb{R}$ is consistent with a given edge labelling $z : E \to \mathbb{R}$ if and only if

$$x(v) - x(u) = z(u, v) \quad \forall (u, v) \in E \quad (2)$$

Let us denote the incidence vector of the edge (u, v) with

$$b_{(u,v)} = (0, \dots, \underset{\uparrow u}{-1}, \dots, \underset{\uparrow v}{1}, \dots, 0)^\top \quad (3)$$

Let x be the vector containing all the node labels and z the vector containing the edge labels (ordered as in B). Equation (2) can be written as

$$x^\top b_{(u,v)} = z(u, v) \quad \forall (u, v) \in E \quad (4)$$

Let B be the $n \times m$ incidence matrix of G, which has the $b_{(u,v)}$ as columns; it is easy to see that for all the edges the equation above becomes $x^\top B = z^\top$, or

$$B^\top x = z. \quad (5)$$

If the graph is connected, the rank of B is $n - 1$, hence there are infinitely many solutions. This is consistent with the fact that the synchronisation problem is defined up to a constant, so one can w.l.o.g. arbitrarily set $x_k = 0$ for a chosen $k \in V$. Removing x_k from the unknowns and the corresponding row in B leaves a full-rank $n - 1 \times m$ matrix B_k (also called "reduced" incidence matrix). Hence we obtain the least-squares solution:

$$x = (B_k B_k^\top)^{-1} B_k z. \quad (6)$$

C. SYNCHRONISATION IN $(\mathbb{IR}, +)$

Stepping from real numbers to interval, synchronisation can be defined as follows:

Definition II.1 (interval consistency)**.** A (sharp) node labelling $x : V \to \mathbb{R}$ is interval-consistent with a given edge labelling $\mathbf{z} : E \to \mathbb{IR}$ if and only if

$$x(v) - x(u) \in \mathbf{z}(u, v) \quad \forall (u, v) \in E \quad (7)$$

Definition II.2 (interval synchronisation problem)**.** Given an edge labelling $\mathbf{z} : E \to \mathbb{IR}$, the solution to the interval synchronisation problem is the node labelling $\mathbf{s} : V \to \mathbb{IR}$ defined as the set of all the interval-consistent node labelling, i.e.:

$$\mathbf{s} = \{x : V \to \mathbb{R} \mid x \text{ is interval consistent} \wedge x(k) = 0\} \quad (8)$$

for a chosen k as in (6).

A straightforward enclosure of the solution can be obtained by extending (7) to interval values $\mathbf{x} \in \mathbb{IR}^n, \mathbf{z} \in \mathbb{IR}^m$, so:

$$\mathbf{x} = (B_k B_k^\top)^{-1} B_k \mathbf{z}. \quad (9)$$

As in any IA problem, the quality of solution depends on its tightness, and we aim at improving it with respect to this baseline solution.

III. METHOD

We tackle the problem of interval synchronisation as defined above, i.e., the edges are labelled with intervals representing differences between adjacent nodes and the goal is to recover the unknown node labels, that are intervals as well. Since the solution is not unique, because adding a constant to the nodes does not change the differences, synchronisation requires one node label to be taken as reference. We will call this node anchor and set its label to zero (arbitrarily).

The baseline approach would be to simply solve Eq. 9, however this solution i) entails a centralised approach that is not always feasible e.g. in a sensor network, and ii) produces larger enclosures than our method, as we shall see in Sec. IV.

We adopt a distributed, iterative approach, where each node evaluates its label only on the basis of its neighbourhoods. Each node periodically communicates its label to the adjacent nodes; there is no constrain about the timing of this communication, and it is not required that the label is sent to all the adjacent nodes at once. By iterating these steps the node labels asymptotically converge to the interval solution $\mathbf{s}$. The procedure is summarised in Algorithm III.1. In the following we switch to a subscript notation, where $\mathbf{x}_i$ is the state of node i.

ALGORITHM III.1 DISTINTSYNCH

Input: Graph with interval edge-labels $\mathbf{z}_{ij}$
Output: Interval node-labels $\mathbf{x}_i$

Step 0: (anchor) select a node as the *anchor*;

Step 1: (initialisation): all the node labels, except the anchor, are set to $[-\infty, +\infty]$, whereas the anchor label is set to $[0, 0]$;

Step 2: (label propagation): a random node i sends an update to a random neighbour j. The label $\mathbf{x}_i$ is combined with the measure of the edge z_{ij} to provide an estimate of the receiver's label $\hat{\mathbf{x}}_j$:

$$\hat{\mathbf{x}}_j = \mathbf{z}_{ij} + \mathbf{x}_i \tag{10}$$

Step 3: (label update): the label of node j is updated to the intersection between the previous label $\mathbf{x}_i$and the received label $\hat{\mathbf{x}}_j$.

$$\mathbf{x}_j = \hat{\mathbf{x}}_j \cap \mathbf{x}_j; \tag{11}$$

Step 4: (iterate) go to step 2.

Please note that, since the intersection with $[\infty, +\infty]$ leaves the label unchanged, and all the nodes but the anchor are initialised to $[\infty, +\infty]$, only updates originating from the anchor are effective.

A. CORRECTNESS

The property that must be verified is that the labelling produced by Algorithm III.1 contains the solution $\mathbf{s}$, as defined in (8). Inductively, let us assume that the current labelling $\mathbf{x}$ contains the solution and let us see that this property is preserved when node j changes its label upon receiving an update from node i.

First we prove that the estimate $\hat{\mathbf{x}}_j$, computed with (10), contains the solution $\mathbf{s}_i$. The inductive hypothesis is

$$\mathbf{s}_i \subseteq \mathbf{x}_i. \tag{12}$$

From the interval consistency it follows that:

$$\mathbf{s}_j \subseteq \mathbf{s}_i + \mathbf{z}_{i,j}, \tag{13}$$

and by the property of inclusion isotonicity of IA [15] we get

$$\mathbf{s}_i + \mathbf{z}_{i,j} \subseteq \mathbf{x}_i + \mathbf{z}_{i,j} = \hat{\mathbf{x}}_j, \tag{14}$$

hence the thesis:

$$\mathbf{s}_j \subseteq \hat{\mathbf{x}}_j. \tag{15}$$

The current state in turn contains the solution by the inductive hypothesis, hence their intersection contains the solution. At the beginning all the nodes but the anchor are set to infinity, hence they contain the solution by construction. Since the update rule is such that the interval radius is always reduced, Algorithm III.1 converges to a solutions of minimum radius.

B. ANCHOR SELECTION

In some real-world applications the set of nodes is structured so that it is natural to choose a special node as anchor, e.g., the gateway in a sensor network. Most of the times, however, the anchor can be freely chosen because all the nodes are equally important. For the sake of developing the intuition, let us assume that the graph is a tree. Each node label is computed by propagating the anchor label through a single path from the anchor to the node, and the interval radius is monotonically increasing. The radius of the resulting interval depends i) on the number of edges that are traversed and ii) on the interval radius of the corresponding labels. Therefore one would like the anchor to minimise the weighted distance to all the other nodes, where the weights are the radius of the edge labels.

This idea is captured by the notion of closeness centrality (or closeness) of a node, i.e., the inverse sum of the distance from a node to all other nodes in the graph. Hence, to minimise the overestimation of IA, the node with the maximum weighted closeness (where the path length is computed as the sum of the radius of the edge labels) is selected as anchor.

The anchor must be unique: if more than one node has the same weighted closeness value, the tie can be broken, e.g.. with a unique node identifier.

Figure 1 shows a simple graph with symmetrical (for simplicity) interval on the edge labels. Node distances and

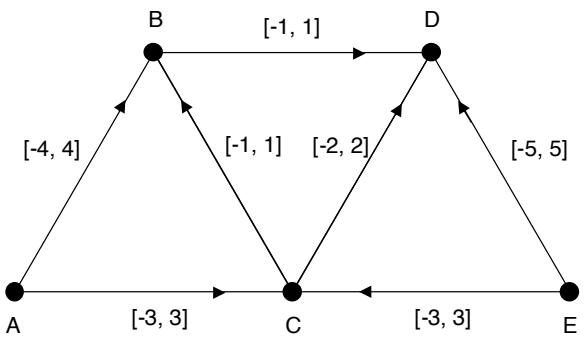

FIGURE 1. Sample graph

TABLE 1. All node pairs distances and centrality for the graph of Figure 1

	A	B	C	D	E	centrality
A	0	8	6	10	16	0,031
B	8	0	14	2	8	0,039
C	6	14	0	16	10	0,027
D	10	2	16	0	6	0,037
E	16	10	10	6	0	0,030

weighted closeness values are reported in Table 1. The maximum of unweighted closeness is achieved by node C, being only one hop away from all other nodes. On the other hand, the maximum of weighted closeness is achieved by node B. Table 2 shows the node labels resulting from every possible anchor selection, assuming that the anchor label is always $[0, 0]$. It turns out that node B is the best anchor choice since it produces the lowest values for both the average and maximum interval radius of the node labels in the solution.

TABLE 2. Interval synchronisation on the sample graph: each row corresponds to a different anchor selection.

anchor	A	B	C	D	E	avg	max
A	[0,0]	[4,-4]	[3,-3]	[5,-5]	[8,-8]	8	16
B	[4,-4]	[0,0]	[7,-7]	[1,-1]	[4,-4]	6.4	14
C	[3,-3]	[7,-7]	[0,0]	[8,-8]	[5,-5]	9.2	16
D	[5,-5]	[1,-1]	[8,-8]	[0,0]	[3,-3]	6.8	16
E	[8,-8]	[4,-4]	[5,-5]	[3,-3]	[0,0]	8	16

C. ALGORITHM WITH ANCHOR ELECTION

In the previous section, it was argued that the node with the maximum weighted closeness should be selected as the anchor. However, this choice requires centralised knowledge of the network, which would violate our assumptions. In other words, since the nodes have no global knowledge of the graph topology, they do not know which node is the anchor (not even the anchor itself). Therefore, we modified the original procedure so that the anchor is elected by the nodes in a distributed way (Algorithm III.2) using the Bellman-Ford algorithm [4].

Before the synchronisation process starts each node evaluates its weighted closeness with the Bellman-Ford distributed algorithm and saves this value to a node attribute that we call tag c_i. The Bellman-Ford distributed algorithm keeps running in the background to keep the weighted closeness values up to date in the face of possible changes of the topology.

Initially, all labels are set to $[0, 0]$ (each node "believes" to be the anchor). During the synchronisation process, the tag is sent to the neighbours along with the node label (modified by the values on the edges). The update rule is such that the nodes are always tagged with the weighted closeness of the node from which the value stored at the node has been propagated.

When a node receive a label estimate from another node with a tag higher than its own it discharges its current label,

ALGORITHM III.2 DISTINTSYNCH W ANCHOR ELECTION

Input: Graph with interval edge-labels $\mathbf{z}_{ij}$

Output: Interval node-labels $\mathbf{x}_i$

Step 0: (weighted closeness) initialise weighted closeness $c_i = \infty \ \forall i$ and run the Bellman-Ford algorithm to calculate the weighted closeness values at all nodes;

Step 1: (initialisation): $x_i = [0, 0] \ \forall i$;

Step 2: (label propagation): a random node i sends an update to a random neighbour j. The label $\mathbf{x}_i$ is combined with the label of the edge $\mathbf{z}_{ij}$ to provide an estimate of the receiver's label $\hat{\mathbf{x}}_j$:

$$\hat{\mathbf{x}}_j = \mathbf{z}_{ij} + \mathbf{x}_i$$

Step 3: (label update):

$$\mathbf{x}_j = \begin{cases} \mathbf{x}_j \text{ if } c_i < c_j \\ \hat{\mathbf{x}}_j \text{ if } c_i > c_j \\ \hat{\mathbf{x}}_j \cap \mathbf{x}_j \text{ if } c_i = c_j \end{cases} \tag{16}$$

$$c_j = \max(c_i, c_j) \tag{17}$$

Step 4: (iterate) go to step 2.

because the information received for the update come from a better origin (and it becomes "aware" of not being the anchor). This way, the labels coming from origins different from the anchor are progressively overwritten, leaving only the labels synchronised with the anchor (see Algorithm III.2).

Please note that, with respect to Algorithm III.1, the initialisation of nodes' labels to $[-\infty, +\infty]$ is implicitly here, for any time a node's label is overwritten because of the test on c, this can be interpreted as the intersection between the received estimated label and $[-\infty, +\infty]$.

D. DISTRIBUTED BELLMAN-FORD

In this section we will briefly review the distributed version of the Bellman-Ford algorithm, which is used for the distance-vector routing protocols in computer networks to find the best routing path to send data packets between routers. The algorithm proceeds by relaxation, in which approximations to the correct distance are replaced by better ones until they eventually reach the solution. Each node builds a table of distances to all other nodes, called distance vector, which is sent to adjacent nodes every time a change occurs. The vectors are initially empty and are updated with the only information a node initially knows, i.e., its own identifier, and are sent to the neighbours. Upon receiving an update from a neighbour i, node j merges its vector with the vector of i increased of the cost of edge (i, j) keeping the minimum distance. Eventually the algorithm converges to the true distances. A synchronous version of this algorithm where in one iterative step all the nodes compute their distance vectors at the same time and then exchange them, converges in a number of steps equal to the length of the longest path between any two nodes in the

network. The "distance" metric in Bellman-Ford algorithm is general, and this makes it useful for our purposes. In fact, using the weighted closeness values as metric it becomes a distributed algorithm for the anchor election.

IV. EXPERIMENTS

To evaluate the performance of the algorithm we generated 6 sets of 50 random graphs each. The graphs are connected and planar. The sets differ for number of nodes (20 or 100) and number of edges, labelled with randomly chosen intervals having maximum radius of 10 and average radius of 5. Table 3 reports the statistics, including the cyclomatic number defined as (#edges - #nodes + 1). This is the relevant parameter to the statistics because it measure in a way the connectivity of the graph, and we generated graphs with a cyclomatic number in a wide range of values that well represent real world situations, from loosely to strongly connected networks.

The graphs are available on the public domain at (removed for blind review).

TABLE 3. Relevant statistics of the graphs used in our simulations

set	#nodes	average #edges	std dev #edges	min cyclomatic	max cyclomatic	avg cyclomatic
1	20	28.1	3.7	2	18	9.1
2	20	55.1	7.3	22	50	36.1
3	20	98.9	12.4	50	109	79.9
4	100	418.4	29.2	248	401	319.4
5	100	1379.4	72.4	1128	1456	1280.4
6	100	2560.7	117.8	2207	2710	2461.7

The proposed distributed synchronisation algorithm has been implemented and simulated in Octave[1] using the interval package[2]. For each graph the results have been compared with

- the baseline solution obtained by solving the linear system (9) with IA;
- the solution obtained by propagating the anchor's label along the graph's minimum spanning tree [5], where the cost associated to each edge is the radius of the interval.

Table 4 reports the average values of maximum and mean radius for each set. For each group of the 300 simulations the table reports the average values; however, the distributed synchronisation algorithm performed better than both other approaches not only in average, but in every single simulation too. This is not surprising, since the interval linear system solution suffers from the well-known overestimation and wrapping effect [15], while the spanning tree solution does not exploit loop-closing which indeed is essential in our approach to improve the solution.

Figure 2 shows the typical behaviour of the distributed synchronisation algorithm. The graph used for this simulation has 20 nodes and a cyclomatic number of 36. The grey section of the plot represents the evolution of the algorithm while each node propagates its state through the graph (therefore increasing each node label interval radius) but not all the nodes have been updated at least once with a value propagating from the anchor. This occurs after 215 simulation steps (the dashed vertical line in the figure), when the anchor value has been propagated 111 times. It represents the first valid solution found by the algorithm, whose average and maximum interval radius are 10.8 and 25.2, respectively, smaller than the solution of the linear system solvers (average: 14.5, maximum: 22.7). However, as shown in the graph, the distributed algorithm keeps improving and reaches a stable final solution with average radius of 4.7 and maximum radius of 10.8.

TABLE 4. Interval radius (avg/max) achieved by three algorithms on graphs with different sizes and cyclomatic number (see Tab. 3)

		average cyclomatic number					
		20 nodes			100 nodes		
		9.08	36.12	79.88	319.44	1280.42	2461.7
avg interval	linear system	17.1	14.81	12.24	32.66	18.38	13.94
	spanning tree	13.77	7.16	4.24	11.68	3.92	2.68
	distributed	12.34	5.64	2.7	8.22	1.83	1.29
max interval	linear system	32.14	23.3	16.53	52.69	25.6	17.89
	spanning tree	28.6	15.41	8.12	25.33	7.81	5.04
	distributed	27.44	13.48	5.76	19.67	4.26	2.72

Figures 3 and 4 shows how the three approaches (linear system solver, spanning tree, and distributed synchronisation) produce better results when the graph's cyclomatic number increases. This is somehow expected, since the cyclomatic number is related to the amount of cycles in the graph and therefore to its degree of connectivity. In particular, a higher cyclomatic number implies shorter paths from the the spanning tree root to the leaves, which improves the solution. Our distributed approach, in addition, benefits from the presence of cycles that provides multiple paths over which independent propagation occur.

It also is interesting to notice that the superiority of the

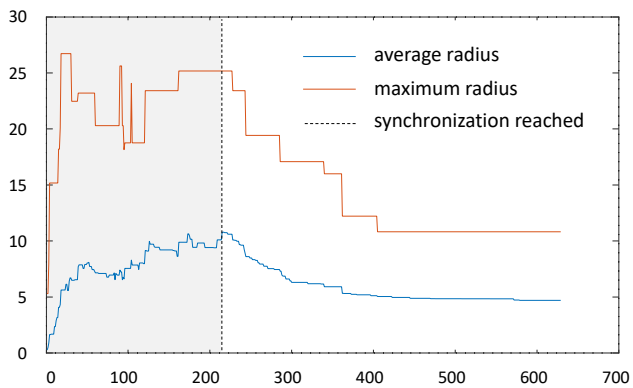

FIGURE 2. Evolution of the interval radius (max/avg) during a simulation on the 20 nodes graph.

[1] https://octave.sourceforge.io/
[2] https://octave.sourceforge.io/interval/package_doc/

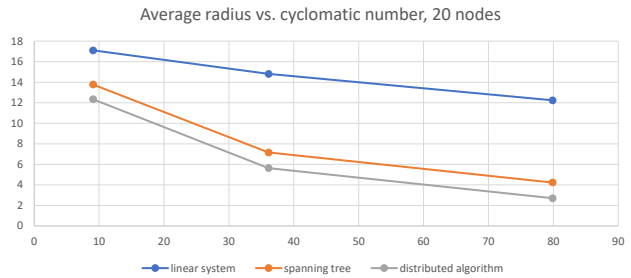

FIGURE 3. Average radius vs. cyclomatic number for the graphs sets of 20 nodes

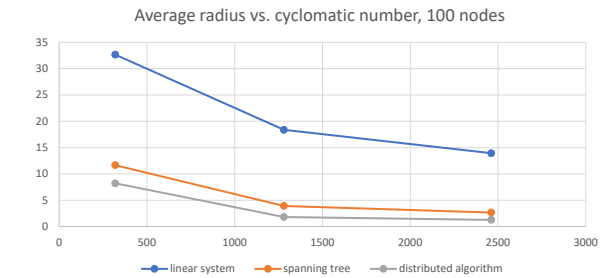

FIGURE 4. Average radius vs. cyclomatic number for the graphs sets of 100 nodes

proposed algorithm is greater for larger graphs with relatively few edges (smaller cyclomatic numbers).

V. CONCLUSION

In this paper, we addressed the problem of graph synchronisation that consists in exploiting pairwise measurements associated with the edges of a graph to obtain consistent values associated to nodes.

We proposed a rigorous method to deal with uncertainty, which we modelled as intervals, and developed an asynchronous, distributed algorithm that converges to the interval solution representing all possible measurement-consistent sharp solutions. Simulations confirm the properties of the proposed approach. These features make it particularly suitable for real-world applications in wireless sensors networks.

REFERENCES

[1] A. Ageev, D. Macii, and D. Petri. Synchronization uncertainty contributions in wireless sensor networks. In Instrumentation and Measurement Technology Conference Proceedings, pages 1986 – 1991, 06 2008.
[2] J. M. Anderson and E. M. Mikhail. Surveying Theory and Practice. WCB/McGraw-Hill, 1998.
[3] F. Arrigoni and A. Fusiello. Synchronization problems in computer vision with closed-form solutions. International Journal of Computyer Vision, 128:26–52, 2020.
[4] R. Bellman. On a routing problem. Quarterly of Applied Mathematics, 16:87–90, 1958.
[5] B. Bollobas. Modern Graph Theory. Springer, 1998.
[6] D. N. Brito, F. L. Pádua, and A. P. Lopes. Using geometric interval algebra modeling for improved three-dimensional camera calibration. Journal of Mathematical Imaging and Vision, 61(9):1342–1369, 2019.
[7] M. Farenzena and A. Fusiello. Rigorous Computing in Computer Vision. In M. Chantler, editor, Vision, Video, and Graphics (2005). The Eurographics Association, 2005.
[8] D. Fontanelli and D. Macii. Master-less time synchronization for wireless sensor networks with generic topology. In 2012 IEEE International Instrumentation and Measurement Technology Conference Proceedings, pages 2785–2790. IEEE, 2012.
[9] A. Giridhar and P. Kumar. Distributed clock synchronization over wireless networks: Algorithms and analysis. Proceedings of the IEEE Conference on Decision and Control, pages 4915–4920, 2006.
[10] D. Grimaldi and F. Lamonaca. Measurement techniques to assess the time synchronization in distributed systems. In Proceedings of the 16th IMEKO TC4 Symposium, Florence, Italy, pages 480–485, 2008.
[11] R. Kearfott, M. Nakao, A. Neumaier, S. Rump, S. Shary, and P. Van Hentenryck. Standardized notation in interval analysis. Vychislitel'nye Tekhnologii, 15, 2010.
[12] D. Malyshev, L. Rybak, G. Carbone, T. Semenenko, and A. Nozdracheva. Optimal design of a parallel manipulator for aliquoting of biomaterials considering workspace and singularity zones. Applied Sciences, 12(4):2070, 2022.
[13] J.-P. Merlet. Interval analysis and robotics. In Robotics research, pages 147–156. Springer, 2010.
[14] R. E. Moore. Interval Analysis. Prentice-Hall, 1966.
[15] R. E. Moore, R. B. Kearfott, and M. J. Cloud. Introduction to interval analysis. SIAM, 2009.
[16] Z. Niu, H. Zhu, X. Huang, A. Che, S. Fu, S. Meng, and Z. Han. Uncertainty quantification method for elastic wave tomography of concrete structure using interval analysis. Measurement, 205:112160, 2022.
[17] R. T. Rajan and A.-J. van der Veen. Joint ranging and synchronization for an anchorless network of mobile nodes. IEEE Transactions on Signal Processing, 63(8):1925–1940, 2015.
[18] W. Russel, D. Klein, and J. Hespanha. Optimal estimation on the graph cycle space. IEEE Transactions on Signal Processing, 59(6):2834 – 2846, 2011.
[19] E. Serpedin and Q. M. Chaudhari. Synchronization in Wireless Sensor Networks: Parameter Estimation, Performance Benchmarks, and Protocols. Cambridge University Press, 2009.
[20] T. Surmacz, B. Wojciechowski, M. Nikodem, and M. Słabicki. Distributed time management in wireless sensor networks. In Proceedings of the Ninth International Conference on Dependability and Complex Systems (DepCoS-RELCOMEX), pages 443–453. Springer, 2014.
[21] R. Voges and B. Wagner. Timestamp offset calibration for an imu-camera system under interval uncertainty. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, October 2018.
[22] B. Xue, Z. Li, P. Lei, Y. Wang, and X. Zou. Wicsync: A wireless multi-node clock synchronization solution based on optimized uwb two-way clock synchronization protocol. Measurement, 183:109760, 2021.
[23] X. Xue, H. Qin, and H. Lu. High-precision time synchronization of kinematic navigation system using gnss rtk differential carrier phase time transfer. Measurement, 176:109132, 2021.
[24] Y. Zhou, S. Zhou, G. Hao, and J. Zhang. Bridge influence line identification based on big data and interval analysis with affine arithmetic. Measurement, 183:109807, 2021.

ANDREA FUSIELLO received the Laurea (M.S.) degree in computer science from the University of Udine (Italy) in 1994, and the Dottorato di Ricerca (Ph.D.) degree in computer engineering from the University of Trieste, in 1999. In 1999 he was a Research Fellow with the Heriot-Watt University, Edinburgh. From 2001 to 2011, he was with the Department of Computer Science, University of Verona. As an Associate Professor he joined the DPIA at the University of Udine in 2012, and became Full Professor in 2023. He is the author of more than 150 articles, and holds two patents. His research interests covers various topics in Computer Vision, Photogrammetry and Image Analysis, with a focus on 3-D modelling/reconstruction. He is an Associate Editor of IEEE Transactions on Image Processing.

Prof. Fusiello was a recipient of the Best Paper - Honorable Mention award at the International Conference on Computer Vision 2021.

PIER LUCA MONTESSORO was born in Torino (Italy) in 1961. He received the Dr. Eng. degree in Electronic Engineering from the Polytechnic of Turin, Italy, in 1986, and now he is full professor in Computer Science at University of Udine, Italy. Previously he has been with the Italian National Council for Scientific Research in Italy and scientific consultant for the Digital Equipment Corporation (later Compaq) in Mass. (USA) in the field of simulation for VLSI design. His research interests, after several years spent on CAD systems for digital circuits design and on multimedia systems for e-learning, are currently focused on computer networks, ICT security and pervasive computing, in particular distributed controls and algorithms for agents-based systems. He has been chair and organizer of the WCC 2013 workshop "International Workshop on Cloud Convergence: challenges for future infrastructures and services", hosted in the IEEE ICC conference and chair of the 30th edition of the Didamatica conference, held in Udine.