AN EFFICIENT ALGORITHM FOR THE SYNTHESIS OF FULLY CONSISTENT GRAPHS

Martin Kreißig and Bin Yang

Institute of Signal Processing and System Theory, University of Stuttgart
email: {martin.kreissig, bin.yang}@iss.uni-stuttgart.de

ABSTRACT

In this paper we present an efficient algorithm for the synthesis of fully consistent graphs. A consistent graph is a graph whose cyclic sum of edge weights along all loops is zero. It plays an important role in many sensor array processing applications like Time Difference of Arrival (TDOA) based source localization. By applying the concept of fundamental loops, a linearly independent basis of the loop space of the graph, our algorithm is able to find all consistent sets of edge weights for the full graph efficiently.

Index Terms— synthesis of consistent graphs, TDOA based localization, difference measurements, sensor fusion

1. INTRODUCTION

We consider sensor array applications where differences $w_{ij} = u_i - u_j$ of sensor measurements u_i and u_j are processed. They inherit the property that the sum of these difference measurements along any loop is zero: $w_{ij} + w_{jk} + \ldots + w_{li} = u_i - u_j + u_j - u_k + \ldots + u_l - u_i = 0$. This relation is known as the *zero cyclic sum*. We can abstract a set of sensors to vertices of a graph and the difference measurements to weights of edges between vertices. If such a graph fulfills the zero cyclic sum condition, we call it a *consistent graph*.

The concept of consistent graphs can be seen in many applications like electrical network theory, sensor fusion and TDOA based source localization [1]. For the latter example, this concept has been already exploited by the DisAmbiguation of TDOA Estimation in Multipath, Multisource environments (DATEMM) algorithm, which synthesizes consistent graphs to reduce the complexity for realtime speaker localization [2]. In this scenario, the difference measurement is the TDOA of the speech signal between a pair of microphones. The TDOAs that stem from one speaker create a consistent graph. Hence if we find all consistent graphs, we reduce the number of false localizations. DATEMM turned out to be an efficient algorithm, but it has two disadvantages. First it starts the synthesis of consistent graphs by triples (loops containing three edges) which do not always exist in an incomplete graph. Secondly, it may lose solutions if multiple sources exist. These restrictions make DATEMM unfeasible for general applications.

In this paper, we present a new efficient algorithm for the synthesis of *fully consistent* graphs. Based on the concept of fundamental loops, the term fully consistent graphs means that all edges have to be assigned a consistent edge weight. In real applications one also has to consider *partial consistency*, where only subgraphs are consistent. Fig. 1 shows a graph with 5 vertices v_i and 7 edges with the corresponding edge weights w_i. It is easy to verify that the subgraph consisting of the four vertices v_1, v_2, v_3, v_4 is consistent because the cyclic sum of edge weights w_i along all loops in the subgraph (e.g. l_1 and l_2) is zero. The full graph consisting of all vertices and edges is, however, not consistent because the cyclic sum of edge weights along loop l_3 is not zero. Hence the graph in Fig. 1 is only partially consistent. The synthesis of partially consistent graphs is a much more challenging problem and will be addressed in future.

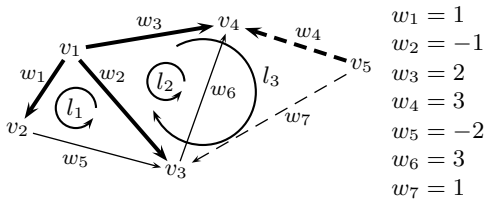

Fig. 1. A graph which is only partially consistent (without v_5).

In the next section we introduce the concept of consistent graphs. In Sec. 3, we present our efficient algorithm to obtain them and discuss its complexity. In Sec. 4, we introduce the notation of approximate consistency. In Sec. 5, we evaluate the theoretical discussion on the complexity given some generated data.

Throughout this paper, we use the following notations: Matrices are bold, vectors underlined and unordered sets are indicated by blackboard bold.

2. CONSISTENT GRAPH

For the subsequent synthesis we denote a graph by $G(\mathbb{V}, \mathbb{E})$ defined by its vertex set $\mathbb{V} = \{v_1, \ldots, v_M\}$ which represents the sensors and the edge set $\mathbb{E} = \{e_1, \ldots, e_N\}$ that indicates whether a value has been measured between a pair of sensors. $\mathbb{W}_n = \left\{w_1^{(n)}, \ldots, w_{K_n}^{(n)}\right\}$ is the weight set for the edge e_n containing different measurements. These weight sets span the complete search space $\mathbb{W} = \mathbb{W}_1 \times \cdots \times \mathbb{W}_N$ from which we aim to find all consistent assignments $\underline{w} = [w_1, \ldots, w_N]^T \in \mathbb{W}$.

The graph is complete if for each pair of vertices there exists an edge in $\mathbb{E}$. It is connected if there exists at least a path between all pairs of vertices. A spanning tree is a subgraph of G that reaches all vertices without closing any loop. The edge set of a spanning tree is denoted by $\mathbb{T}_s$. For any spanning tree, we obtain one fundamental loop (FL) by adding an edge from the complementary tree $\mathbb{T}_c = \mathbb{E} \setminus \mathbb{T}_s$ to it. Each FL can be represented by an $N \times 1$ vector $\underline{l}_i$ containing 0 if the specific edge is not included in the loop, 1 if the edge and loop point to the same direction and -1 otherwise. There are $N - M + 1$ FLs that are represented by the FL matrix $\mathbf{B}_f = [\underline{l}_i]$ [1]. For the graph of Fig. 1, one possible spanning tree is given by the thick edges, thus leading to the complementary tree

$\mathbb{T}_c = \{e_5, e_6, e_7\}$ and to the following FL matrix

$$\mathbf{B}_f = \left[\begin{array}{lrrr} \text{loops} \rightarrow & l_1 & l_2 & l_3 \\ \text{edges} \downarrow & & & \\ \hline e_1 & 1 & 0 & 0 \\ e_2 & -1 & 1 & -1 \\ e_3 & 0 & -1 & 1 \\ e_4 & 0 & 0 & -1 \\ e_5 & 1 & 0 & 0 \\ e_6 & 0 & 1 & 0 \\ e_7 & 0 & 0 & 1 \end{array}\right]. \quad (1)$$

From [3] we know that for a given graph G, $\mathbf{B}_f$ represents a linearly independent basis of all loops. This enables us to check the cyclic sum condition by

$$\mathbf{B}_f^T \underline{w} = \underline{0}. \quad (2)$$

3. AN EFFICIENT SYNTHESIS ALGORITHM

Our algorithm to obtain all consistent solutions for the full graph $G(\mathbb{V}, \mathbb{E})$ with weight set $\mathbb{W}$ performs such that we first determine a spanning tree $\mathbb{T}_s$. This can be done either by Depth-First Search (DFS) or Breadth-First Search (BFS). We obtain the FLs by adding the edges from the complementary tree to the spanning tree.

Then we compute the consistent solutions for each FL separately. A FL can contain three or more edges depending on the given graph and the choice of the spanning tree. Finally we merge these consistent FLs together to fully consistent graphs in the sense of (2). In the following, we discuss the details of the algorithm.

3.1. Spanning tree

In the graph theory, there exist several algorithms to obtain a spanning tree in a connected graph. The most familiar ones are BFS and DFS [4]. Both algorithms need an initial vertex to start their search.

BFS searches all neighbours of the current vertex and marks them as visited if this has not been done previously. Next, one of the newly marked neighbours is defined as the new reference vertex and its neighbours are explored. If there is no new vertex to be visited, BFS steps back to the previous reference vertex and looks for the next neighbour which has not been explored yet and marks it as a reference one. This leads to a rather wide spanning tree, which has small distances from the initial vertex to all others.

In comparison, DFS leads to a rather narrow and deep spanning tree. DFS searches for a neighbour of the inital vertex, marks it as visited and defines it directly as the new reference vertex. Then it explores the neighbours of the new reference vertex. If there is an unvisited one, it is defined as the new reference. Otherwise, DFS steps back to the previous reference vertex and explores its next unvisited neighbour.

BFS and DFS are known to find their solution in $\mathcal{O}(M+N)$ steps. The worst case is when they have to explore each vertex and each edge once.

For our purpose it is better to apply the BFS algorithm as it produces shorter FLs. For the same reason, it is recommended to choose the initial vertex of the spanning tree algorithm as the one with most neighbours.

3.2. Fundamental loops

As stated previously, we determine FLs by adding edges of the complementary tree $\mathbb{T}_c$ to the spanning tree $\mathbb{T}_s$. The loops are closed by an algorithm that searches through all edges of the spanning tree. Thus the complexity for determining the FL matrix $\mathbf{B}_f$ can be bounded by $\mathcal{O}((N-M)M)$.

3.3. Consistent fundamental loops

Now we search all consistent weight assignments for each FL. This is done by a simple brute-force search of all weight combinations for each FL. The consistent solutions for each FL are saved.

The number of edges per FL is between 3 and M. If a graph is complete and we use a BFS spanning tree, all FLs will have only 3 edges. In the worst case, one FL can contain at maximum M edges (all $M-1$ edges from $\mathbb{T}_s$ and one edge from $\mathbb{T}_c$) if we choose a DFS spanning tree. Hence the complexity for determining all consistent solutions of all FLs is between $(N-M)K^3$ and $(N-M)K^M$ if all edge weight sets $\mathbb{W}_n$ have the same cardinality $K_n = |\mathbb{W}_n| = K$. We see here that a BFS tree is clearly better than a DFS one. Below we assume that after this step, the i-th FL has finally $\bar{K}_i$ $(\leq K)$ consistent solutions.

3.4. Loop merging

Starting with the sets of consistent solutions for all $N-M+1$ FLs, we now merge them to fully consistent graphs. The idea is straightforward. We start with all $\bar{K}_1$ consistent solutions of the 1st FL and merge all $\bar{K}_2$ consistent solutions of the 2nd FL to them. So we have to check $\bar{K}_1 \cdot \bar{K}_2$ combinations. One combination is valid if there is no conflict between the consistent solutions of both FLs. A conflict arises if both FLs share one or several common edges and the weights of these edges are different in both FLs. The merging is successful if a) the weights of the common edges are identical in both FLs or b) both FLs have no common edges. As a result, we obtain $\bar{K}_1\bar{K}_2$ or less consistent subgraphs containing both FLs. In that way, we merge iteratively the consistent solutions of the remaining FLs to them. This is illustrated in Fig. 2 where the graph shown in a) has three FLs denoted as l_1, l_2, l_3. Each of them is assumed to have two consistent solutions. These consistent FLs shown in b) are denoted by $l_{o,p}$ where o indicates the loop number and p the index of its consistent solution. In c) we present the loop merging process where $\times$ denotes a conflict in the edge weights of common edges, which is the case e.g. for $l_{1,1}$ and $l_{2,1}$ on edge (v_1, v_2). The solid arrows indicate a valid merging and the dashed arrows show skipped combinations.

It is clear that only these pairs of consistent FLs lead to a reduction in the combinations if they have common edges. Thus it is recommended to sort the FLs in such a way that FLs having a large number of common edges are merged first.

The complexity of the loop merging step depends on the number of consistent solutions. If we assume $\bar{K}_i = \bar{K}$, we obtain at most $\bar{K}^{N-M+1}$ different combinations in the tree of Fig. 2. It is easy to see that this is a very pessimistic bound, because a lot of combinations will be discarded at early stages due to conflict. In the best case, if the merging of two FLs always returns $\bar{K}$ consistent solutions out of $\bar{K}^2$ combinations, the overall complexity of the merging step has the order $\mathcal{O}\left((N-M)\bar{K}^2\right)$.

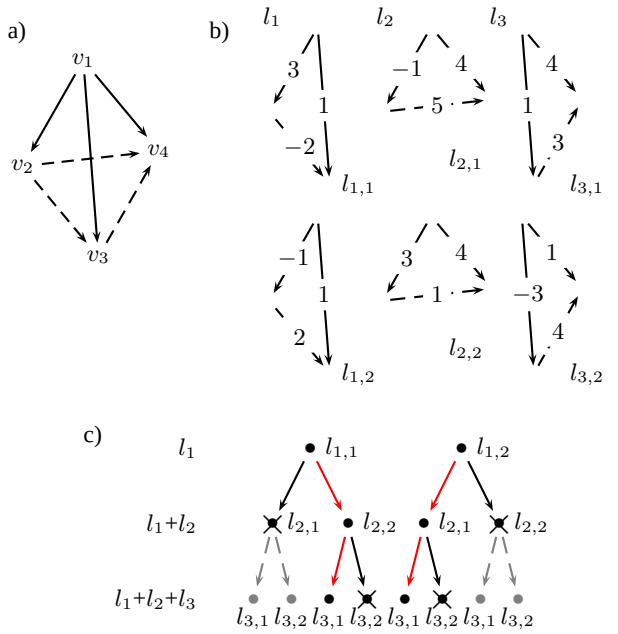

Fig. 2. Illustration of loop merging for three fundamental loops with each two consistent solutions.

3.5. Complexity

The different steps of our synthesis algorithm require a computational complexity of $\mathcal{O}(M+N)$ for the spanning tree, $\mathcal{O}((N-M)M)$ for obtaining the fundamental loops, $\mathcal{O}\left((N-M)K^3\right) - \mathcal{O}\left((N-M)K^M\right)$ for finding consistent fundamental loops and $\mathcal{O}\left((N-M)\bar{K}^2\right) - \mathcal{O}\left(\bar{K}^{N-M+1}\right)$ for loop merging. Hence the complexity C can be upper bounded by $\mathcal{O}((N-M)K^M + \bar{K}^{N-M+1})$ in the worst case and lower bounded by $\mathcal{O}((N-M)K^3 + (N-M)\bar{K}^2) \approx \mathcal{O}\left((N-M)K^3\right)$ if all FLs are triples. This shows a significant improvement since the brute-force approach would check all K^N combinations of the edge weights with typically $K \gg \bar{K}$.

3.6. Comparison to previous algorithms

This algorithm differs to DATEMM due to its clear structure and its flexibility in terms of FLs. In DATEMM consistent triples (loops of three edges and vertices) are synthesized and merged if two triples have the same weight on a common edge. This combined subgraph is appended by another triple if there are common edge weights and so on until the maximal connected graph is synthesized. Therefore the set of consistent triples must be scanned several times. This implies high cost in complexity. The new algorithm compares all pairs of consistent loops only once and is hence simpler. Moreover it is always possible to find a set of FLs in any connected graph. In contrast, DATEMM requires triples which may not exist in a connected graph while DATEMM does not.

In [5] we already presented a similar synthesis algorithm which is based on a Back-Tracking (BT) search on the FLs. The main disadvantage of BT is that only full consistent solutions are accepted, i.e. a consistent weight assignment to all edges. Instead our new algorithm is able to return consistent partial solutions provided by the consistent FLs and combinations of them. Note that there are partial consistent solutions that are not covered by the FLs. Additionally, in [5] a consistent assignment like $l_{2,2}$ in Fig. 2 is computed at run time after $l_{1,1}$. When the same assignment is reached in another branch like $l_{1,2}$ in our example it is recalculated. In the new algorithm we compute the consistent assignments only once and that leads to a lower complexity.

4. APPROXIMATELY CONSISTENT GRAPHS

So far we considered the ideal case of perfect zero cyclic sum. In real applications, the edge weights w_n like TDOA measurements are never precise due to noise, measurement error and time discretization. Hence, the cyclic sum of even matching edge weights is normally small, but not exactly zero. This is called an approximately consistent graph. In this case, we have to replace the exact consistency condition (2) by a relaxed one. This can be done individually for all FLs $\underline{l}_i$

$$|\underline{l}_i^T \underline{w}| \leq \delta \quad \forall i \tag{3}$$

or for all FLs together

$$||\mathbf{B}_f^T \underline{w}||_p \leq \Delta \tag{4}$$

where $||\cdot||_p$ is a suitable p-norm. The three p-norms $p=1$ (the sum of magnitude of all elements of $\mathbf{B}_f^T \underline{w}$), $p=2$ (the euclidean norm of $\mathbf{B}_f^T \underline{w}$), and $p=\infty$ (the maximum magnitude of all elements of $\mathbf{B}_f^T \underline{w}$) are worth to be studied. The threshold value δ or Δ have to be adjusted according to the variance of the edge weights w_n. In addition, δ and Δ can also be chosen according to the length (number of edges) of the FLs for uncorrelated edge weight errors with equal variance.

5. SIMULATIONS

We implemented our algorithm in C++. To perform a feasible verification of the computational complexity in Sec. 3, we choose a complete graph with $M=5$ vertices, $N=10$ edges and generate $\bar{K}$ different fully consistent weight vectors $\underline{w}_k$ satisfying (2). In the first experiment, no additional erroneous edge weights are generated and hence the total number of weights per edge is $K=\bar{K}$. We apply the brute force approach and the new synthesis algorithm with either a DFS or a BFS spanning tree to find all $\bar{K}$ fully consistent graphs. This experiment is repeated 50 times and the average run time in seconds is plotted in Fig. 3 over $K=\bar{K}$. We see a significant reduction in run time of our algorithms compared to the brute force approach. The improvement is even bigger when we use a BFS spanning tree due to shorter FLs. The solid line in Fig. 3 shows a scaled version of the computational complexity $(N-M)K^3$. We see that it matches the run time of our algorithm very well. Similar results were also obtained when running the simulations on graphs with more vertices and complete as well as non-complete but connected graphs.

In the second experiment, we checked the impact of wrong measurements on the run time. We generated $K=10$ weights per edge where $\bar{K}=1,\ldots,5$ weights are consistent and $K-\bar{K}$ weights are randomly chosen and thus wrong. In Fig. 4 we present the run time of our synthesis algorithm based on either a BFS or a DFS spanning tree. We see that the run time is constant and does not depend on $\bar{K}$. This verifies the dependence of the complexity on K rather than $\bar{K}$ as predicted in Sec. 3.5.

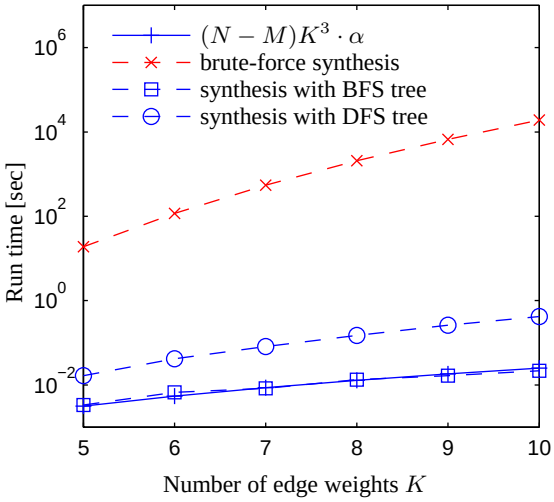

Fig. 3. Comparison of the run time of the brute force and the new synthesis algorithms with either a BFS or DFS spanning tree for a complete graph of $M = 5$ vertices. $\alpha = 5 \cdot 10^{-6}$ is a scaling factor between the computational complexity in terms of the number of operations and the run time in seconds.

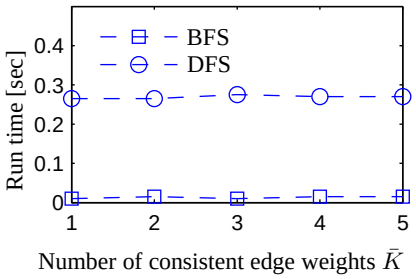

Fig. 4. Run time of our synthesis algorithm with either a BFS and DFS spanning tree over $\bar{K}$ consistent out of $K = 10$ edge weights ($M = 5, N = 10$).

In the final simulation, we consider the effect of errors in the edge weights on the synthesis of approximately consistent graphs. For simplicity, we model the edge weight errors as an additive white Gaussian noise with the variances $\sigma^2 = [0.2, 0.4, 0.6]$. The edge weights $w_k^{(n)}$ are randomly chosen in the interval $[-500; 500]$ and we generated a complete graph of $M = 5$ and $N = 10$.

We generated $\bar{K} = 10$ consistent solutions for each (σ^2, δ) pair, where δ is given in (3). In Fig. 5, we plot the number of originally generated consistent solutions that our synthesis algorithm has found (true positive) over the number of new consistent solutions (true negative). The additional graphs occur due to the relaxation of the cyclic sum condition in (3). The threshold δ for checking approximate consistency is chosen to be $[0.75, 1.0, 1.25, 1.5, 1.75]$ where smaller δ values correspond to the lower points in Fig. 5 and larger δ values to upper points.

As expected we observe a strong increase of the number of additional solutions for large δ. We also see that it is impossible to obtain all desired solutions without synthesizing new ones. This effect is well known from the detection theory that the false alarm probability increases with the detection probability.

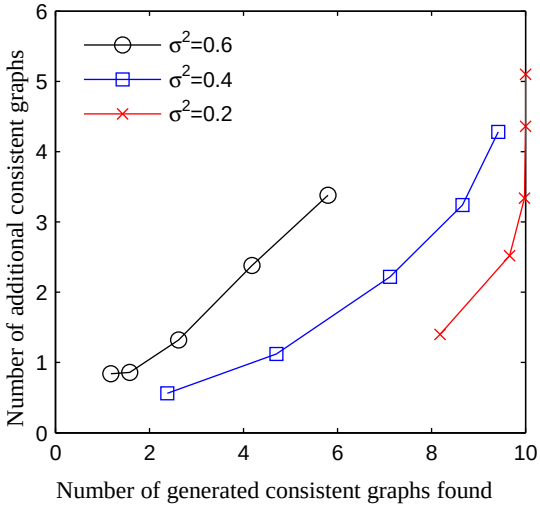

Fig. 5. The number of consistent graphs found in a complete graph with $M = 5$ vertices and given $\bar{K} = 10$ consistent graphs, for each setup (σ^2, δ). The lowest points are related to $\delta = 0.75$ and the highest ones of each line to $\delta = 1.75$ (with a step size of 0.25).

6. CONCLUSION

The synthesis of consistent graphs can be applied to many sensor fusion applications to reduce their complexity. Here we have presented a new efficient algorithm that first finds consistent fundamental loops and then merges them to fully consistent graphs.

Two open questions remain to be solved. First, if a given graph is not connected, there is no spanning tree and our algorithm can not be applied. In this case, we need some preprocessing to decompose a disconnected graph into connected components before applying our synthesis algorithm to each component. Second, there is no efficient synthesis algorithm yet for partially consistent graphs.

REFERENCES

[1] Bin Yang and Martin Kreißig, "An introduction to consistent graphs and their signal processing applications," in *Proc. IEEE ICASSP*, 2011, pp. 2740 –2743.

[2] Jan Scheuing and Bin Yang, "Disambiguation of TDOA estimation for multiple sources in reverberant environments," *IEEE Trans. on ASLP*, vol. 16, no. 8, pp. 1479–1489, Nov. 2008.

[3] N. Balabanian and T. A. Bickart, *Electrical Network Theory*, John Wiley & Sons, 1969.

[4] Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, Pearson Education, 2 edition, 2003.

[5] Martin Kreißig and Bin Yang, "Efficient synthesis of consistent graphs," in *Proc. EUSIPCO*, 2010, pp. 1364–1368.