EFFICIENT SYNTHESIS OF CONSISTENT GRAPHS

Martin Kreißig and Bin Yang

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

ABSTRACT

A consistent graph is a graph with zero cyclic sum of weights of edges along all loops. Given a number of possible weights for each edge, we study the problem of synthesizing consistent graphs, i.e. to find the appropriate combinations of weights, which form consistent graphs. This problem plays an important role in, e.g. source localization based on time difference of arrival (TDOA). By using the concept of loop matrix known from the electric network theory, we propose some novel systematic approaches for the efficient synthesis of consistent graphs. We describe our algorithms, demonstrate their performance and compare their computational complexity, both in theory and in experiments.

1. INTRODUCTION

A consistent graph contains vertices, edges and weights of edges whose sum along any closed path (loop) is zero.

A well known example of consistent graphs is the voltage graph of an electric circuit where the weight is the voltage between two nodes. According to Kirchhoff's second law, the sum of voltages along any loop is zero. This kind of consistent graphs also plays an important role in many other scientific and technical problems. In source localization based on TDOA, for example, a (generalized) cross-correlation between two sensor signals is frequently used to estimate the TDOA values. Unfortunately, the cross-correlation typically shows many peaks due to desired direct path and (disturbing) multi-path propagation and multiple sources.

The callenge is to distinguish between the TDOA values caused by direct or multi-path propagation and to find those TDOA values from different sensor pairs that are related to the same source. In [1, 2], Scheuing and Yang proposed for the first time the concept of consistent TDOA graphs for solving this problem. Each sensor represents a vertex and each TDOA value corresponds to a weight of an edge, respectively. It was observed in [1, 2] that the sum of TDOA values along any loop in the TDOA graph must be zero if all these TDOA values stem from the same source and direct path propagation. Hence the task is now to synthesize consistent TDOA graphs, given a number of possible TDOA estimates for each sensor pair.

To our knowledge, this graph synthesis problem has never been addressed systematically in the literature. Scheuing and Yang proposed a first algorithm DATEMM (disambiguation of TDOA estimates in multi-path multi-source environments) in [1, 2] for this purpose. Its basic idea is to first look for consistent TDOA triples (three vertices) and then, using a bottom-up approach, combine them to consistent TDOA quadrupels (four vertices), consistent star graphs (quadrupels sharing a common initial triple) and finally complete consistent TDOA graphs [2]. But this algorithm is ad-hoc and has not been completely understood yet in the multiple source case. Also it is open whether there are more efficient synthesis approaches than DATEMM.

The purpose of this paper is to study the efficient synthesis of consistent graphs in a theoretical and systematical way. By using the well known concept of loop matrix from the electric network theory, we develop different novel approaches to synthesize consistent graphs. We show their performance and compare their computational complexity.

2. CONSISTENT GRAPH

A graph $G(V,E)$ is defined by a set of M vertices (nodes) $V = \{v_i, \cdots, v_M\}$ and a set of N edges $E = \{e_1, \cdots, e_N\}$. Clearly, $N \le \binom{M}{2} = \frac{1}{2}M(M-1)$, because there are at most $\binom{M}{2}$ vertex pairs. The graph is *directed* if its edges are specified by a start vertex i and an end vertex j. It is *weighted* if a weight $x(e) \in \mathbb{R}$ is assigned to each edge e. In addition, we assume that the weight changes its sign if we change the direction of the edge, i.e. $x_{ij} = -x_{ji}$. This is pretty much like the voltage between two nodes in an electric circuit.

A connection of neighboured edges is called a *path*. If the start and end vertex of this path are the same, the path is closed and hence a *loop*. When there exists a pairwise connection of all vertices by a path, the graph G is *connected*. If a connected graph has $N = |E| = \binom{M}{2}$ edges for $M = |V|$ vertices, it is *complete* because all pairs of vertices are connected by one edge.

In a directed weighted graph, one can compute the sum of all weights along any loop in a certain direction. This is called the *cyclic sum* of weights for that loop. A graph is *consistent* if this sum is zero for all loops in the graph. Fig. 1 shows a complete consistent graph with $M = 5$ vertices and $N = 10$ edges. The number on the edges represent the weights.

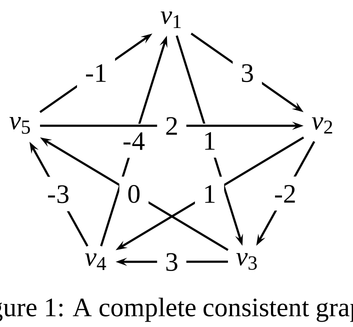


Figure 1: A complete consistent graph

Note that for a given localization problem with known sensor and source positions, we obtain a unique set of TDOA values between different sensor pairs which form a consistent TDOA graph. Conversely, we can easily show by a constructive proof (not given in the paper) that for any given set of consistent edge weights as in Fig. 1, it is also possible to construct a geometric setup[1] whose TDOA values match the

[1]In fact, there is an infinite number of geometric setups.

edge weights. In other words, each set of consistent weights has its localization counterpart. For this reason, we use in this paper simple integer edge weights for illustration without considering the geometrical setups of the corresponding localization problems.

3. INCIDENCE AND LOOP MATRIX

A formal description of the connectivity of a graph is given by the *incidence matrix* [3], which describes the relationship between the vertices and edges. For the graph in Fig. 1, the incidence matrix is

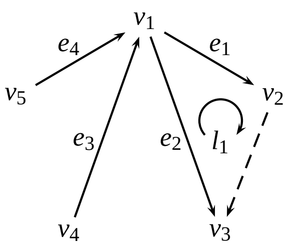
$$\mathbf{A} = \left[\begin{array}{c|cccccccccc} \text{edges}\rightarrow & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 & e_9 & e_{10} \\ \text{vertices}\downarrow & & & & & & & & & & \\ \hline v_1 & 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ v_2 & -1 & 0 & 0 & 0 & 1 & 1 & -1 & 0 & 0 & 0 \\ v_3 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 1 & 1 & 0 \\ v_4 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & -1 & 0 & 1 \\ v_5 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & -1 & -1 \end{array}\right]. \tag{1}$$

Start vertices are marked with a “1” and end vertices by a “-1”.

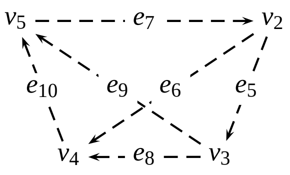
In this paper, we study connected graphs only, i.e. each vertex is connected to each other vertex over a path. A necessary condition is that the graph has at least $N = M - 1$ edges. According to [4], it is then always possible to sort the columns of $\mathbf{A}$ such that its first $M-1$ columns are linearly independent. We rewrite $\mathbf{A}$ as

$$\mathbf{A} = [\mathbf{A}_{\text{st}}\ \mathbf{A}_{\text{ct}}] \tag{2}$$

where $\mathbf{A}_{\text{st}}$ is an $M \times (M-1)$ matrix with full rank $M-1$ and $\mathbf{A}_{\text{ct}}$ is an $M \times (N-M+1)$ matrix. $\mathbf{A}_{\text{st}}$ represents the *spanning tree* and $\mathbf{A}_{\text{ct}}$ the *complementary tree*, also *cotree* of the graph, respectively [4]. The spanning tree defines a subgraph of G that reaches every vertex without closing any loop. One possible spanning tree of the graph in Fig. 1 is shown in Fig. 2(a) with solid lines.



(a) Spanning tree

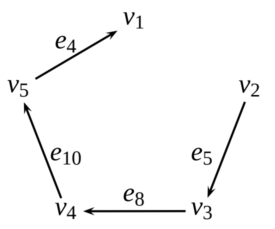


(b) Complementary tree

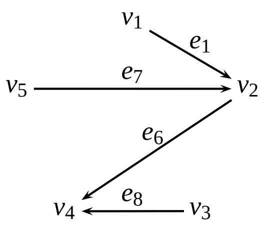
Figure 2: A possible spanning tree of the graph in Fig. 1 and its complementary tree

The complementary tree to Fig. 2(a) is presented in Fig. 2(b) and contains the twigs. Each of the twigs closes one *fundamental loop* with the spanning tree, also referred to as *elementary cycle* in graph theory. Fundamental loops are the minimum set of loops in G without any redundancy. Each column in $\mathbf{A}_{\text{ct}}$ is hence responsible for one fundamental loop. In Fig. 2(a) it is shown how the twig e_5 from the complementary tree closes one loop l_1 with the spanning tree.

There are different ways to obtain the spanning tree from the incidence matrix $\mathbf{A}$. A breadth-first search would primarily look in the surrounding of a vertex like in Fig. 2(a). A deepening search would try to find the longest sequence of vertices and may produce the spanning tree of Fig. 3(a). Fig. 3(b) shows another possible spanning tree.



(a) Spanning tree of the deepening search



(b) Another spanning tree

Figure 3: Other possible spanning trees for the example of Fig. 1

The *loop matrix* describes the relationship between the edges and fundamental loops. For the graph in Fig. 1, we use the six twigs $e_5, e_6, \cdots, e_{10}$ of Fig. 2(b) to form six fundamental loops $l_1, l_2, \cdots, l_6$ with the spanning tree in Fig. 2(a). The corresponding loop matrix is

$$\mathbf{B} = \left[\begin{array}{c|cccccccccc} \text{edges}\rightarrow & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 & e_9 & e_{10} \\ \text{loops}\downarrow & & & & & & & & & & \\ \hline l_1 & 1 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ l_2 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ l_3 & -1 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ l_4 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ l_5 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ l_6 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]. \tag{3}$$

A “1” indicates identical edge and loop direction, while a “-1” reflects opposite edge and loop direction.

The loop matrix $\mathbf{B}$ has the dimension $(N-M+1) \times N$. It has the full rank $N-M+1$ as apparent from the identity matrix in the last columns in (3). The identity matrix can always be achieved for any loop matrix when the fundamental loops in (3) appear in the same order as the twigs. For this reason, $\mathbf{B}$ can be written as

$$\mathbf{B} = [\mathbf{B}_{\text{st}}\ \mathbf{I}_{N-M+1}] \tag{4}$$

where $\mathbf{B}_{\text{st}}$ contains the first $M-1$ columns of $\mathbf{B}$ corresponding to the edges in the spanning tree. Clearly, $N \geq M$ if a connected graph has at least one loop.

4. CHECK OF CONSISTENCY

The fundamental loops play an important role for the synthesis of consistent graphs. A necessary and sufficient condition for a graph to be consistent is that all $N-M+1$ fundamental loops are consistent, i.e. have a zero cyclic sum of weights. It is no longer necessary to prove the consistency of all loops.

Let x_i be the weight assigned to the edge e_i $(1 \leq i \leq N)$. We define the edge weight vector as $\mathbf{x} = [x_1, \cdots, x_N]^T \in \mathbb{R}^N$. The consistency of all fundamental loops and hence of the whole graph can be easily checked by

$$\mathbf{B} \cdot \mathbf{x} = \mathbf{0}. \tag{5}$$

The main problem, however, is that each edge e_i has a number of possible weights. They could be the different TDOA values for a sensor pair caused by different sources and direct as well as multi-path propagations. Let W_i be the set of $S_i = |W_i|$ weights $x_i \in W_i$ for edge e_i. Clearly, there is a total number of $\prod_{i=1}^{N} S_i$ possibilities for the weight vector $\mathbf{x} \in (W_1 \times \cdots \times W_N)$. Each of these vectors corresponds to a graph with a certain combination of weights whose consistency has to be examined according to (5). This process is called synthesis of efficient graphs. In the next section, we propose different approaches for this purpose.

5. SYNTHESIS ALGORITHMS

In this section, we present systematic approaches for the efficient synthesis of consistent graphs given sets of weights W_i $(1 \le i \le N)$. For an easier estimation of the computational complexity, we assume an equal number of weights per edge $S_i = S$. All synthesis algorithms rely on the use of the loop matrix $\mathbf{B}$ in (4).

5.1 The brute force approach

The brute force approach tries all $\prod_{i=1}^{N} S_i = S^N$ possible weight vectors $\mathbf{x} \in (W_1 \times \cdots \times W_N)$. For each combination $\mathbf{x}$ of weights, the consistency condition (5) has to be examined. Without taking the zero entries in $\mathbf{B}$ into account, the computational complexity of this consistency check is $(N-M+1)N$ operations, where one operation is either an addition/subtraction or comparison. This complexity can be reduced by considering the zero entries in $\mathbf{B}$. The price is a higher complexity in program code and address calculations. The total computational complexity of the brute force approach is hence

$$C_{\text{BruteForce}}(M,N,S) = S^N \cdot (N-M+1)N. \tag{6}$$

5.2 Kirchhoff Potential Synthesis (KiPoS)

KiPoS exploits the special structure of the loop matrix in (4) that is always possible for a connected graph. We partition $\mathbf{x}$ into two subvectors of length $M-1$ and $N-M+1$, i.e. $\mathbf{x} = [\mathbf{x}_{\text{st}}^T \, \mathbf{x}_{\text{ct}}^T]^T$. $\mathbf{x}_{\text{st}} = [x_1, \cdots, x_{M-1}]^T$ contains the weights of the spanning tree while $\mathbf{x}_{\text{ct}} = [x_M, \cdots, x_N]^T$ contains the weights of the twigs of the cotree. By using (4), the consistency condition can be written as

$$\mathbf{x}_{\text{ct}} = -\mathbf{B}_{\text{st}} \cdot \mathbf{x}_{\text{st}}. \tag{7}$$

Instead of considering all S^N combinations for $\mathbf{x}$, we now look at $\prod_{i=1}^{M-1} S_i = S^{M-1}$ combinations for $\mathbf{x}_{\text{st}}$. For each of these combinations of the spanning tree, we compute $\mathbf{B}_{\text{st}} \cdot \mathbf{x}_{\text{st}}$ and compare it to the weights of the cotree $\mathbf{x}_{\text{ct}}$. Since we need $M-2$ additions/subtractions and S comparisons for each of the $N-M+1$ rows of $\mathbf{B}_{\text{st}} \cdot \mathbf{x}_{\text{st}}$ (fundamental loops), the computational complexity of this algorithm is reduced to

$$C_{\text{KiPoS}}(M,N,S) = S^{M-1} \cdot (N-M+1) \cdot (M-2+S). \tag{8}$$

This algorithm has a nice interpretation in terms of the Kirchhoff voltage law. We assign a zero “electrical potential” to a reference vertex and $M-1$ “potentials” to the remaining vertices. This assignment corresponds to the choice of $\mathbf{x}_{\text{st}}$. Then we compute the differences of the potentials and compare them to the “voltages” $\mathbf{x}_{\text{ct}}$ along the twigs of the cotree. For this reason, this synthesis algorithm is called *Kirchhoff Potential Synthesis (KiPoS)*.

5.3 KiPoS with initial pruning (KiPoS-P)

In order to further reduce the computational complexity, we propose to apply a pruning technique before we perform the previous algorithm. The basic idea of pruning is to reduce the size of the weight sets $S_i = |W_i|$. In particular, we are highly interested to reduce S_i $(1 \le i \le M-1)$ for the spanning tree since these numbers affect the complexity of the KiPoS significantly, see the term S^{M-1} in (8).

We propose to prune the weight sets by checking the consistency of each of the $N-M+1$ fundamental loops. Assume that the fundamental loop l_j contains $M \ge N_j \ge 3$ edges. Then we obtain S^{N_j} possible combinations of weights for this loop. The consistency of each weight combination can be examined by $N_j - 1$ operations. The complexity of the pruning step of $N-M+1$ fundamental loops is thus

$$C_{\text{Prune}}(M,N,S) = S^{N_j}(N_j-1)(N-M+1). \tag{9}$$

If all fundamental loops are triples with $N_j = 3$ which is always possible for complete graphs with a breadth-first search spanning tree like in Fig. 2(a), we obtain $C_{\text{Prune}}(M,N,S) = S^3 2(N-M+1)$.

After the pruning step, we have removed all weights from W_i which are not consistent for any fundamental loop and thus cannot contribute to the consistency of the whole graph. The reduced weight set for edge e_i is $\overline{W}_i \subset W_i$ with $\overline{S}_i = |\overline{W}_i| \le S_i$. Then we apply the previous KiPoS algorithm to find consistent graphs. This algorithm is called Kirchhoff Potential Synthesis with Pruning (KiPoS-P). Its overall complexity is

$$C_{\text{KiPoS-P}}(M,N,S,\overline{S}) = C_{\text{Prune}}(M,N,S) + C_{\text{KiPoS}}(M,N,\overline{S}) \tag{10}$$

if $\overline{S}_i = \overline{S}$ for all $1 \le i \le N$.

5.4 Depth-First Search with Back-tracking (DFS-BT)

The theory of obtaining a solution to a problem with constraint variables is well known as the *Constraint Satisfaction Problem* (CSP). One example is Sudoku, where we have 9×9 variables. Each of them can take a value from the set $\{1, \cdots, 9\}$. The constraints are that each row, column and 3×3-square can only have a single occurrence of each value, i.e. the union of the values along a row, column or 3×3-square must be the complete set $\{1, \cdots, 9\}$ again.

The theory of CSP goes back to Mackworth in 1977 [6] with the description of *arc-* and *path-consistency*. This means a sequence of variables that are connected by logical or arithmetical constraints that have to be fulfilled. We can map our problem of consistent graphs to such a CSP formulation by interpreting the zero cyclic sum condition (5) as an arithmetic constraint on the weights.

A CSP is mainly solved by a method called *search strategy* [5]. We use the *Depth-First-Search* (DFS) [7] which searches a tree or structure in such a way that all nodes along a path are explored. The ordering of the path in our case is defined by the ordering of the fundamental loops. In the first step, we search the edges in the first fundamental loop and afterwards we explore the edges of the second fundamental loop and so on. In (3), for example, the order of the edges to be explored is $e_1, e_2, e_5, e_3, e_6, e_4, e_7, e_8, e_9, e_{10}$. We assign a weight $x_i^j \in W_i$ to each edge e_i following this order. x_i^j represents the j-th weight in W_i. The result is a tree like in Fig. 4 where each transition from one edge of the graph to another is determined by the weight assigned to the starting edge.

When we follow one path top down, we obtain one complete assignment of all edges. If we check all assignments for consistency, there are $\prod_{i=1}^{N} S_i$ different possibilities like in the brute force approach in 5.1. Therefore, the *back-tracking*

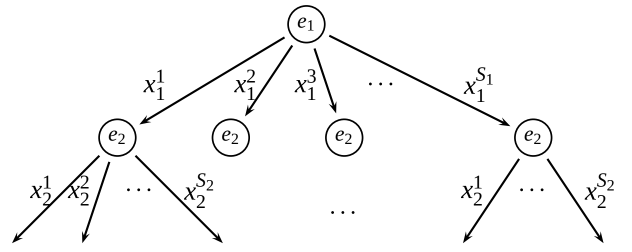


Figure 4: A tree structure showing the process of assignment of weights x_i^j to edges e_i.

(BT) [9] is added to the DFS. Each time the search algorithm detects a failure in a new assignment, it skips this value or returns to the previous edge. The same applies when the algorithm reaches the last edge.

For the synthesis of consistent graphs, a failure is a contradiction to the zero cyclic sum condition in (5), i.e. we find at a certain edge that the new assigned weight does not match to the previously assigned weights in the sense of (5). In this case, we try the next weight from W_i for the current edge and test its consistency to the previous weights. If all weights for the current edge have already been considered, we track back to the previous edge in the search tree and proceed with its next weight. The improvement of this approach to brute force is that we skip complete subtrees in Fig. 4, when the weights of edges of higher levels conflict.

This algorithm is quite similar to the algorithm presented in [2] where triples are used to synthesize a consistent graph. Each triple corresponds to a row in $\mathbf{B}$ (fundamental loop) for complete graphs. The test of the zero cyclic sum condition along a triple in [2] corresponds to the pruning of W_i. The concatenation of rows of $\mathbf{B}$ and the assignment of weights to edges is equivalent to the combination of consistent triples that have a common edge.

The difference is that we have erased the redundancy of considering all triples. Instead we consider only the fundamental loops. Moreover, our approach does not require a complete graph whose fundamental loops can be chosen as triples. The loop matrix is a more general concept and applicable to arbitrary connected graphs.

In general, it is difficult to determine precisely the complexity of the DFS-BT algorithm. Taking the consistency constraints of the fundamental loops into account, we can approximate the complexity as follows. To test the first fundamental loop, we have to assign a weight to all included edges. In the worst case, that means M edges ($M-1$ from $\mathbf{B}_{\text{st}}$ and one from $\mathbf{I}_{\text{N-M+1}}$ in (4)). Hence we obtain S^M different assignments for the first fundamental loop. Assuming that the first fundamental loop has S' consistent weight combinations, we can reduce the number of assignments from S^M to S'. For the next fundamental loop, only one new edge (from $\mathbf{I}_{\text{N-M+1}}$) with S possible weights is added which gives $S \cdot S'$ different assignments to be checked. Subsequently this is done for all $N-M$ remaining fundamental loops. Since each fundamental loop needs at maximum $M-1$ operations for a consistency check, we obtain a complexity of

$$C_{\text{DFS-BT}}(M,N,S,S') = (M-1)\cdot\left[S^M + S\cdot S'\cdot(N-M)\right]. \tag{11}$$

5.5 DFS-BT with initial pruning (DFS-BT-P)

Just like in section 5.3 we can further reduce the computational complexity of DFS-BT by applying an initial pruning step to the weight sets W_i. By testing the consistency of each fundamental loop, we can reduce W_i to $\overline{W}_i$ with $\overline{S} = |\overline{W}_i|$ before we apply the DFS-BT algorithm. This algorithm is called DFS-BT with initial pruning (DFS-BT-P).

In general, $S \geq S' \geq \overline{S}$ because there may exist more consistent fundamental loops than complete consistent graphs including all loops. But for the sake of simplicity we assume $S' = \overline{S}$ consistent graphs and subgraphs (loops). Hence the overall complexity of DFS-BT-P is

$$C_{\text{DFS-BT-P}}(M,N,S,\overline{S}) = C_{\text{Prune}}(M,N,S) + C_{\text{DFS-BT}}(M,N,\overline{S},\overline{S}). \tag{12}$$

6. EVALUATION AND COMPARISON

In this section, we evaluate the proposed synthesis algorithms and compare their complexity.

6.1 Simulation setup

We implemented all algorithms in MATLAB. For a graph with M vertices and N edges, we generated for each edge e_i a set W_i of $S = |W_i|$ weights. We propose the so called potential approach to generate $\overline{S}$ graphs which are consistent by definition: We first assign M arbitrary "electrical potentials" to the M vertices. Then we compute their differences ("voltages") as the weight of the edges x_i. In this way, the resulting weight vector $\mathbf{x}$ always satisfies the consistency condition $\mathbf{B}\cdot\mathbf{x} = \mathbf{0}$. The remaining $S - \overline{S}$ weights per edge are generated randomly.

6.2 More consistent graphs than generated

We applied all algorithms to these simulated weight sets. All algorithms find all $\overline{S}$ consistent graphs as expected. Interestingly, the synthesis algorithms often find more than $\overline{S}$ consistent graphs. Fig. 5 shows such an example. For a simple triple, we generated $S = \overline{S} = 3$ consistent weight vectors $\mathbf{x}_1 = [2,1,1]^T$, $\mathbf{x}_2 = [3,2,1]^T$, $\mathbf{x}_3 = [3,3,0]^T$ as shown in Fig. 5(a). The corresponding weight sets are $W_1 = \{2,3\}$, $W_2 = \{1,2,3\}$, $W_3 = \{0,1\}$. The synthesis algorithms, however, produce in addition to these consistent graphs a new one $\mathbf{x}_{\text{new}} = [2,2,0]^T$ in Fig. 5(b) whose weights are a combination from W_1, W_2, W_3.

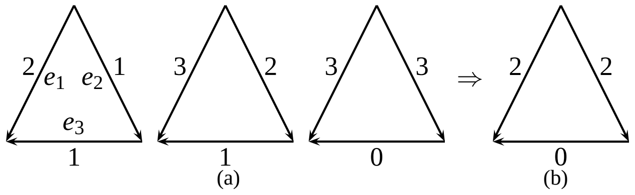


Figure 5: Given the three consistent graphs in 5(a), a new consistent graph in 5(b) occurs by combining existing weights.

The reason for this phenomenon is pretty simple. If all $\mathbf{x}_i$ are consistent in the sense of $\mathbf{B}\cdot\mathbf{x}_i = \mathbf{0}$, then any linear combination of $\mathbf{x}_i$ is also consistent because of $\mathbf{B}\cdot(\sum_i c_i\mathbf{x}_i) = \mathbf{0}$. In the above example, $\mathbf{x}_{\text{new}} = 2(\mathbf{x}_2 - \mathbf{x}_1)$ and all weights in $\mathbf{x}_{\text{new}}$ occured in $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$.

6.3 Comparison of computational complexity

For comparing the complexity of different algorithms, we focused on complete graphs with $M = 6$ and $M = 10$ vertices and $N = \binom{M}{2}$ edges. Fig. 6 shows the theoretical complexities of the brute force approach (6), KiPoS (8) and DFS-BT (11) for a varying number $S = \overline{S}$ of weights per edge. The plots illustrate a polynomial increase of complexity due to $O(S^N)$ in (6) and $O(S^M)$ in (8) and (11). It is obvious that the brute force approach has the highest complexity.

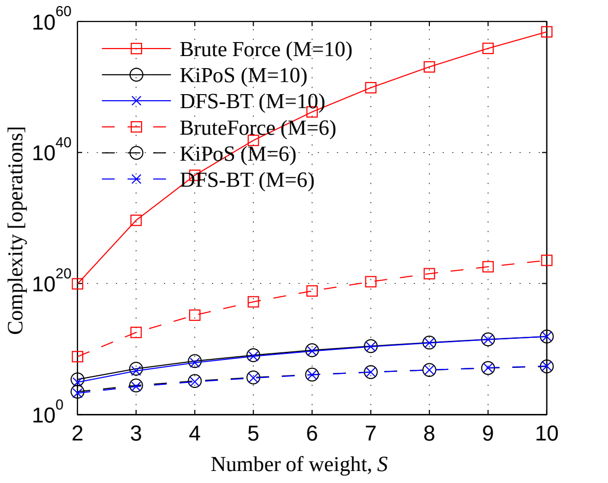


Figure 6: A comparison of the theoretical complexities of the brute force approach, KiPoS and DFS-BT for $M = 10$ (solid) and $M = 6$ (dashed)

Next we discuss the improvement of the initial pruning step for KiPoS and DFS-BT. This step causes little additional operations of order $O(S^3M^2)$ as shown in (9), but reduces the overall complexity significantly due to $\overline{S}^M \ll S^M$ in (10) and (12).

This is shown in Fig. 7 where KiPoS, KiPoS-P, DFS-BT and DFS-BT-P are compared for $M = 6$ vertices, $N = \binom{M}{2} = 15$ edges, a varying number of S weights per edge and $\overline{S} = \frac{1}{2}S$ consistent graphs for each value of S. The solid curves plot the theoretical number of operations according to (8), (10), (11), (12). The dashed curves show the run time of MATLAB simulations in μsec. As expected, the initial pruning step reduces significantly the complexity because not all weights contribute to consistent graphs and are deleted from W_i. Secondly, the theoretical comlexity estimates agree well with the experimental results, at least for $S < 18$. For larger values of S, the complexity in simulations increases faster than the theoretical predictions. This effect is related to the one described in section 6.2. The larger the number of weights per edge is, the higher the probability is that the synthesis algorithms find more consistent graphs than we generated.

7. CONCLUSION AND OUTLOOK

We have presented several systematic algorithms for synthesizing consistent graphs with zero cyclic sum of weights of edges. All algorithms are based on the loop matrix which specifies the fundamental loops in a graph. We not only studied different search strategies, but also developed a simple effective pruning step to reduce the size of the weight sets for

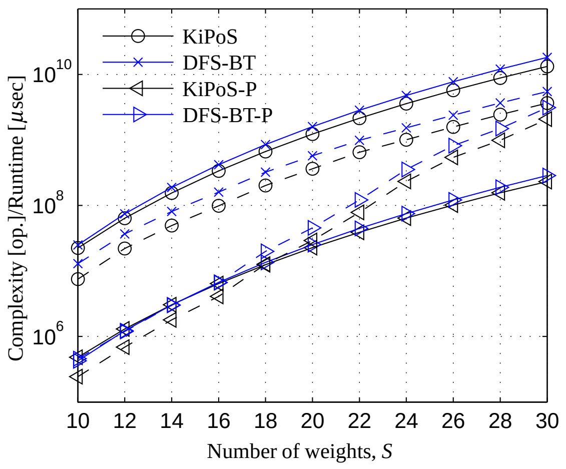


Figure 7: A comparison of the theoretical (solid) and experimental complexities (dashed) of KiPoS(-P) and DFS-BT(-P) for $M = 6$ and $\overline{S} = S/2$

the edges. We derived the computational complexities for all algorithms and verified them in computer simulations.

In the future, we will study how to further reduce the computational complexity of synthesis algorithms and make a comparison to the DATEMM algorithm in [2]. In addition, some practical issues like disconnected graphs and approximately consistent graphs will be investigated.

REFERENCES

[1] Scheuing, J. and Yang, B.,“Disambiguation of TDOA Estimates in Multi-Path Multi-Source Environments (DATEMM)”, in *IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2006, pp. 837–840

[2] Scheuing, J. and Yang, B.,“Disambiguation of TDOA Estimation for Multiple Sources in Reverberant Environments”, in *IEEE Transactions on Audio, Speech and Language Processing*, Nov. 2008, pp. 1479–1489

[3] Jungnickel, D., “Graphs, Networks and Algorithms (Algorithms and Computation in Mathematics)”, *Springer-Verlag*, 2004

[4] Balabanian, N. and Bickart, T.A., “Electrical Network Theory”, *John Wiley & Sons, Inc. (New York)*, 1969

[5] Schulte, C. and Stuckey, P. J., “Efficient Constraint Propagation Engines”, in *Transactions on Programming Languages and Systems* , vol. 31, pp. 2:1–2:43, Dec. 2008.

[6] Mackworth, A. K., “Consistency in networks of relations”, *Artificial Intelligence 8, 1*, pp. 99–118, 1977

[7] Tarjan, R., “Depth-First Search and linear graph algorithms”, *SIAM Journal in Computing*, vol. 1, issue 2, pp. 146–160, 1972

[8] Bang-Jensen, J. and Gutin, G., “Digraphs - Theory, Algorithms and Applications”, *Springer-Verlag*, 2006

[9] Kumar, V., “Algorithms for Constraint Satisfaction: A Survey”, *Artificial Intelligence Magazine*, vol. 13, number 1, pp. 32–44, 1992