DISCUSSION ON THE BOTTOM-UP SYNTHESIS OF CONSISTENT GRAPHS: DATEMM REVISITED

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

In this paper we discuss the synthesis algorithm DATEMM, which finds consistent graphs in a TDOA based speaker localization scenario. We analyse it in terms of complexity and point out the shortcoming of this algorithm, that not all possible solutions are found. Then we present an improved algorithm to fix this problem. Finally we verify our analysis through simulations and give a comparison to other synthesis algorithms.

Keywords: *Efficient Graph Synthesis, Consistent Graphs, Speaker Localization, DATEMM*

1. INTRODUCTION

There are some applications in signal processing where the difference measure $w_{ij} = p_i - p_j$ of two measurements p_i and p_j of sensors i, j plays an important role. Among others we focus on time difference of arrival (TDOA) based speaker localization where the sensors are microphones and the measure $w_{i,j}$ is the TDOA obtained by evaluating the generalized cross-correlation function. As another application we see the sensor fusion in automotive driver assistance systems where the sensors are radar or video and one obtains relative distance and velocity measurements between vehicles. Unfortunately, the measurements can often not be assigned directly to one object, like the speaker or the vehicle to be located, because they are taken from peaks of a signal. The ambiguity increases even more when we take into account the multi-path propagation due to reflections.

Hence we propose a preprocessing step to reduce the combinatorial complexity. This step is based on the simple property that the sum of measures of one target is zero at a set of neighboured sensors:

$$w_{ij} + w_{jk} + \ldots + w_{li} = p_i - p_j + p_j - p_k + \ldots + p_l - p_i = 0 \quad (1)$$

The proposed approach intends to minimize the time, necessary to obtain the set of difference measures for one target, while keeping all solutions.

In this paper we first give a short introduction to the notations we use that stem from graph theory. Then we summarize a known algorithm for the specified problem called DATEMM introduced by [1] and improved in [2] and point out its shortcoming. Finally we propose an improved algorithm to obtain all solutions.

2. GRAPH THEORETICAL DESCRIPTION

A graph $G(\mathbb{V}, \mathbb{E}, \underline{w})$ is given by its vertex set $\mathbb{V} = \{v_1, \ldots, v_M\}$ of $M = |\mathbb{V}|$ elements and edge set $\mathbb{E} = \{e_1, \ldots, e_N\}$ with N elements. The graphs considered here are simple, directed and weighted with weight vector $\underline{w} = [w_1, \ldots, w_N]^T$, $w_n : e_n \to \mathbb{R}$. That means, the graphs have no parallel edges and edges are defined by a start and end vertex, which must be different (no loops on the same vertex).

A complete graph is described by $N_{\max} = \binom{M}{2}$ edges, where each vertex shares an edge with all other vertices.

The combinatorial problem that arises is the assignment of K_n possible values per edge e_n, defined by its weight set $\mathbb{W}_n = \{w_n^1, \ldots, w_n^{K_n}\}$. The brute-force search for those assignments that fulfill (1) requires a test of all $\prod_{n=1}^{N} K_n$ combinations, which is obviously very inefficient. An example of a complete, weighted graph with $M = 6$ vertices and weight set cardinality of $K_n = 1\,\forall n$ is plotted in Fig. 1.

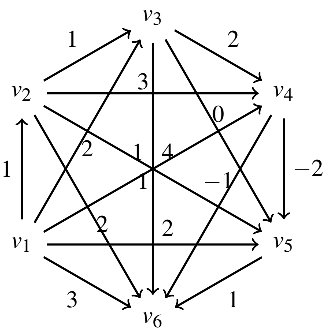


Figure 1*: A complete graph with $M = 6$ vertices and single edge weight.*

In [3, 4] we proposed and discussed an efficient synthesis algorithm to find those assignments that fulfill a zero cyclic sum condition along all loops in the graph. We call these graphs *consistent graphs* in the following. The approach in [3] is based on a fixed topology of the graph, which leads to a set of fundamental loops. These are used to apply Back-Tracking, a known search algorithm from artificial intelligence [5], on the search space $\mathbb{W} = \mathbb{W}_1 \times \cdots \times \mathbb{W}_N$. In this paper we focus on an approach that starts from consistent triples, which means small cycles of three edges that have a zero cyclic sum, and synthesize the complete graph in a bottom-up way.

3. DATEMM REVISITED

3.1 Summary

DATEMM (Disambiguation of TDOA Estimation in Multipath Multisource environments) [1] contains the subsequent steps, to synthesize the consistent graphs in a given setup $G(\mathbb{V}, \mathbb{E}, \underline{w})$, $\underline{w} \in \mathbb{W}_1 \times \cdots \times \mathbb{W}_N$, one by one. A *triple t* is a graph defined by the set of 3 vertices $\mathbb{V}_t = \{v_1, v_2, v_3\}$, 3 edges $\mathbb{E}_t$ defined as the set of directed pairs $\mathbb{E}_t = \{e_n = (u, v) : u, v \in \mathbb{V}, u \neq v\}$ and the corresponding weight vector $\underline{w}_t$, thus $t = (\mathbb{V}_t, \mathbb{E}_t, \underline{w}_t)$. We use $V(t)$ to refer to the vertex set of t, $E(t)$ to reference its edge set and $\underline{w}(t)$ for its weight vector, respectively.

- S1 Find all consistent triples and save them to the triple set $\mathbb{T}$. In Fig. 2 below, the consistent triples composed from the vertices v_1, v_2, v_3, and v_4 of Fig. 1 are shown.
- S2 Take an initial triple, e.g. the vertices $\{v_1, v_2, v_3\}$ (solid lines), and find all neighboured triples. This creates the set $\mathbb{Q}_q$ of consistent *quasi-quadruples*, whose elements are subgrpahs consisting of 4 vertices and 5 edges. In Fig. 3 the quasi-quadruples of Fig. 2 are plotted.
- S3 Merge all quasi-quadruples that share common edge weights to complete, consistent *quadruples* (in the following only quadruples for short). For Fig. 3 we obtain the quadruple of Fig. 4(a).

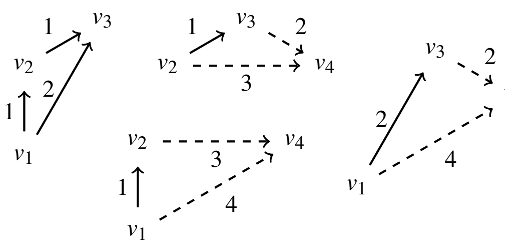


Figure 2*: All possible triples of the subgraph, spanned by the vertices v_1, v_2, v_3, and v_4 from Fig. 1.*

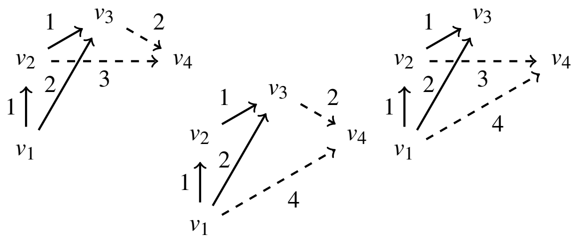
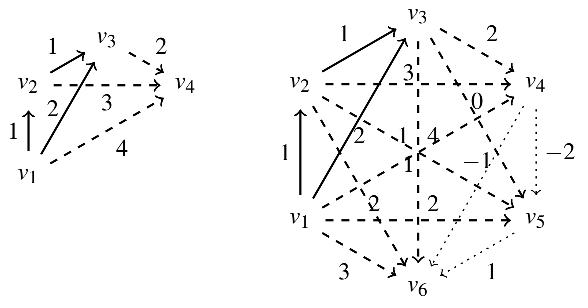


Figure 3*: All possible quasi-quadruples created by the 4 vertices of Fig. 2.*



(a) Quadruple generated by the quasi-quadruples of Fig. 3.

(b) Complete consistent graph generated by concatenating all quadruples that include the initial triple.

Figure 4*: The last two steps to synthesize one complete consistent graph.*

S4 When merging two quadruples that include both the initial triple, there is always an edge missing between vertices, which do not belong to the initial triple. In Fig. 4(b) these edges are plotted as dotted lines. To complete the graph we search in $\mathbb{T}$ for triples that include the missing edge and which do fit with two neighboured edges.

S5 Remove all used triples from the triple set.

S6 If there are still triples left in the triple set, select a new initial triple and go back to S2 to synthesize the next consistent graph.

3.2 Analysis

This synthesis approach is straightforward. The entities at each step represent a consistent subgraph and hence a partial solution to the problem. Remember that the brute-force approach checks all branches of a tree as shown in Fig. 5 for 3 edges. This leads to $\prod_{n=1}^{N} K_n$ possible combinations.

DATEMM divides this tree in blocks having three levels each like in Fig. 5, that represent the three edges of a triple. A block has $\hat{K} \leq K^3$ consistent outputs for $K_n = K$, $(1 \leq n \leq N)$. All these outputs define triples that share the same vertices and edges and differ only in their weights. Thus we introduce the *topological triple* as $\tau = (\mathbb{V}_t, \mathbb{E}_t)$ that defines the underlying edges and vertices. We define $\hat{K}_\tau$ as the number of consistent assignments to the topological triple τ. There exist $T = \binom{M}{3}$ different topological triples in a complete graph. Thus we obtain $\sum_{\tau=1}^{T} \hat{K}_\tau \ll \prod_{n=1}^{N} K_n$ solution branches after the triple generation.

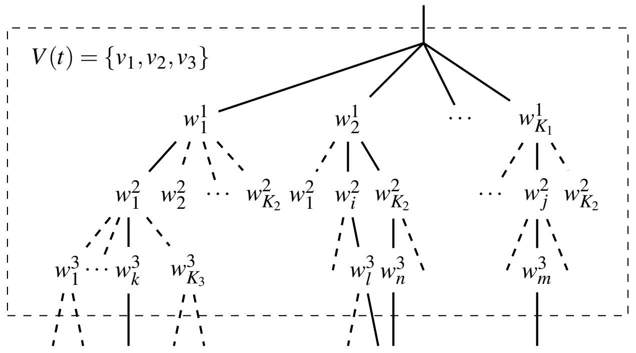


Figure 5*: Weight assignment for a topological triple τ. The dashed lines illustrate the brute-force combinations while the solid lines represent the $\hat{K}_\tau$ consistent assignment of a triple.*

Without any further assumptions, we can only state that $\hat{K}_\tau = K^3$. Due to the concatenation of triples, this number decreases because, given a fixed triple and a topological triple, being added with one common edge, we obtain less than K^2 possible solutions for the merged topological quasi-quadruple, because the common edge is already assigned. Thus the complexity of DATEMM is even smaller: $C_{\text{DATEMM}} < \sum_\tau \hat{K}_\tau$, with $\hat{K}_\tau = K^3$.

DATEMM has been shown to perform very well for speaker localization [1] in terms of efficiency, but it has the shortcoming of not finding all solutions in a given search space $\mathbb{W} = \mathbb{W}_1 \times \cdots \times \mathbb{W}_N$.

3.3 Shortcoming of DATEMM

Having a detailed look at the weight assignment of DATEMM, we can draw a tree like given in Fig. 6 where the step S1 represents the initial triple assignment. In step S2 we add a triple which leads to quasi-quadruples and in S3 we obtain complete quadruples. Then we merge quadruples which are illustrated as a list of triples along a branch in Fig. 6.

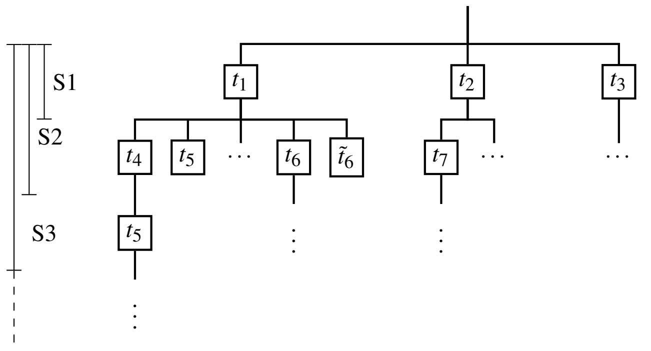


Figure 6*: Triple assignment that leads to a conflict*

Generally, we have an arbitrary number of triples that fit to an initial triple. In our example, the triples t_4, t_5, t_6 and $\tilde{t}_6$ show a common edge (weight) with the initial triple t_1. DATEMM always starts with the most likely triple. Let's assume this is the quasi-quadruple $t_1 \cup t_4$. The $\cup$-operator defines the union of the specific elements of the triples: $t_1 \cup t_4 = G(V(t_1) \cup V(t_4), E(t_1) \cup E(t_4), \underline{w}(E(t_1 \cup t_4)))$.

In S3 we merge with those quasi-quadruples that include t_1, which is $t_1 \cup t_5$ in our case. After the addition of quasi-quadruples we finally add some triples which creates the most left branch in Fig. 6. When the consistent graph steming from t_1 is synthesized, we remove the used triples and proceed with the next initial triple t_2.

Let's assume that $\tilde{t}_6$ has different weights on the same topological triple as t_6. Then $\tilde{t}_6$ will not occur in the branch of t_4. But the removal of triples defined in DATEMM prevents us from finding the graph being synthesized with the combination t_1, t_4, t_5 and $\tilde{t}_6$.

Tab. 1 shows an example of such a circumstance. There are three consistent graphs $\underline{w}_1, \underline{w}_2$ and $\underline{w}_3$ of a graph with $M = 6$ and $N = 15$ (complete). Let's assume that the triple set will lead to $\underline{w}_1$ at first. By excluding these triples from the subsequent synthesis, we prevent the algorithm to find $\underline{w}_3$ completely, which includes the triple $\{v_1, v_2, v_6\}$ with weights $w_1 = 1, w_5 = -2$ and $w_9 = -3$.

	$\underline{w}_1$	$\underline{w}_2$	$\underline{w}_3$
$e_1 = (v_1, v_2) \sim \mathbb{W}_1:$	$\boxed{1}$	2	$\boxed{1}$
$e_2 = (v_1, v_3) \sim \mathbb{W}_2:$	-2	3	1
$e_3 = (v_1, v_4) \sim \mathbb{W}_3:$	1	4	2
$e_4 = (v_1, v_5) \sim \mathbb{W}_4:$	2	5	3
$e_5 = (v_1, v_6) \sim \mathbb{W}_5:$	$\boxed{-2}$	3	$\boxed{-2}$
$e_6 = (v_2, v_3) \sim \mathbb{W}_6:$	-3	1	0
$e_7 = (v_2, v_4) \sim \mathbb{W}_7:$	0	2	1
$e_8 = (v_2, v_5) \sim \mathbb{W}_8:$	1	3	2
$e_9 = (v_2, v_6) \sim \mathbb{W}_9:$	$\boxed{-3}$	1	$\boxed{-3}$
$e_{10} = (v_3, v_4) \sim \mathbb{W}_{10}:$	3	$\textcircled{1}$	$\textcircled{1}$
$e_{11} = (v_3, v_5) \sim \mathbb{W}_{11}:$	4	$\textcircled{2}$	$\textcircled{2}$
$e_{12} = (v_3, v_6) \sim \mathbb{W}_{12}:$	0	0	-3
$e_{13} = (v_4, v_5) \sim \mathbb{W}_{13}:$	1	$\textcircled{1}$	$\textcircled{1}$
$e_{14} = (v_4, v_6) \sim \mathbb{W}_{14}:$	-3	-1	-4
$e_{15} = (v_5, v_6) \sim \mathbb{W}_{15}:$	-4	-2	-5

Table 1: Example of three different consistent graphs that are not all found by DATEMM

4. PROPOSED ALGORITHM EXTENSION

4.1 Change order of comparison

The first improvement that we suggest is the change of the order in which we append the triples to the current subgraph. While DATEMM performs this in a straightforward manner (one compared to all), we apply a rather horizontal scan and compare all triples at stage S2 with each other. Figure 7 illustrates these comparisons as dashed lines (for the synthesis branch with initial triples t_1). This is necessary to find all combinations related to the initial triple at the root of a branch. If we skip this, the first added triple will determine the consistent graph to be found.

We see that all possible combinations are checked in Fig. 7. Then we can remove the triples from the set.

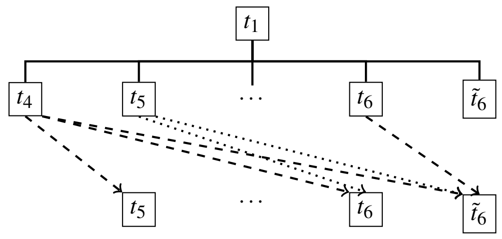


Figure 7*: The new assignment structure is horizontal*

4.2 Introduction of a queue

When checking all combinations of triples and quadruples we still face the problem that the synthesized consistent subgraph may collidate with the current quadruple's triple, as explained in the previous section with triple $\tilde{t}_6$. If we skip this, we would miss the solution including $(t_1 \cup t_4 \cup t_5) \cup \tilde{t}_6$ because t_4 is already processed then.

Therefore we introduce a queue $\mathscr{Q}$ to track those subgraphs that create a new consistent solution branch with a partition of the current subgraph. Each time we detect a collision of edge weights, but the remaining ones match, we append the new weight combination to $\mathscr{Q}$. When the current synthesis is finished, we go back to the queue and process the next element, until the queue is empty. Thus we do not miss a solution.

4.3 Summary

One may point out that the removal of triples in one solution branch excludes its reuse in other graphs. This problem is solved by the circumstance, that a triple with multiple usages has been saved in the quadruple set and can still be used.

Both described improvements, the comparison of all triples against each other that are connected to the initial triple, and the introduction of a queue, enable us to find all consistent graphs.

5. SIMULATIONS

5.1 Setup description

We tested our algorithm whether it finds all solutions. As shown in [4] and explained in Sec. 3.3, we cannot assume a fixed number of consistent graphs in a simulation setup because the superposition of solutions may create new consistent graphs. But to reduce the influence of random, additional, consistent graphs, we apply the following algorithm to generate edge weights. For a given setup of M vertices and K desired consistent graphs:

1. compute random potentials for all vertices
2. calculate difference measures (consistent by definition)
3. compare them to previous weights on each edge
4. if the weights are unique, save them else go to 1
5. while number of graphs $< K$ go to 1

Thus the algorithm finds most likely those graphs defined in the generation, which gives a more objective comparison of the performance with other synthesis algorithms.

The algorithm was implemented in `C++`.

5.2 Verification of the complexity approximation

For $T = \binom{M}{3}$ triples in a complete graph and $K_n = K$ weights that belong to one consistent graph each, we derived in Sec. 3.2 that the number of consistent graphs after step S1 is $T\hat{K}_\tau$. When we assume $\hat{K}_\tau = K$, i.e. K consistent triples for each topological triple, we obtain the bottom line in Fig. 8 as the number of consistent triples. Assuming no further improvements we would obtain K^3 outputs of each topological triple, which is related to the topmost line in Fig. 8. From our measurements we obtained the solid lines, which represent either the number of triples $|\mathbb{T}|_{\text{measured}}$ or quasi-quadruples $|\mathbb{Q}_q|_{\text{measured}}$.

We see that the number of triples fits nearly to $\hat{K}_\tau = K$ for small weight sets and increases a bit for larger weight sets. The number of quasi-quadruples increases similiar to the triples, but with an logarithmic offset. We can also derive from the plot that the number of entities is closer to the theoretical number of triples than to all combinations $\hat{K}_\tau = K^3$, which illustrates the complexity reduction in terms of deleted possible branches in the weight assignment.

The cardinality of the consistent quadruple set and other consistent subgraph sets decreases quickly, as we are merging them instead of adding new possible combinations.

5.3 Validation of the theoretical performance

The improvements mentioned are strongly dependent on the structure of the weight set $\mathbb{W}$, which means how and if the weights create consistent subgraphs. In Fig. 9 we see the runtime t of the new algorithm in seconds, plotted over the number of weights per edge for $M = 5$ and $M = 7$ vertices. Moreover the complexity $C = TK^3$ is plotted in dashed lines.

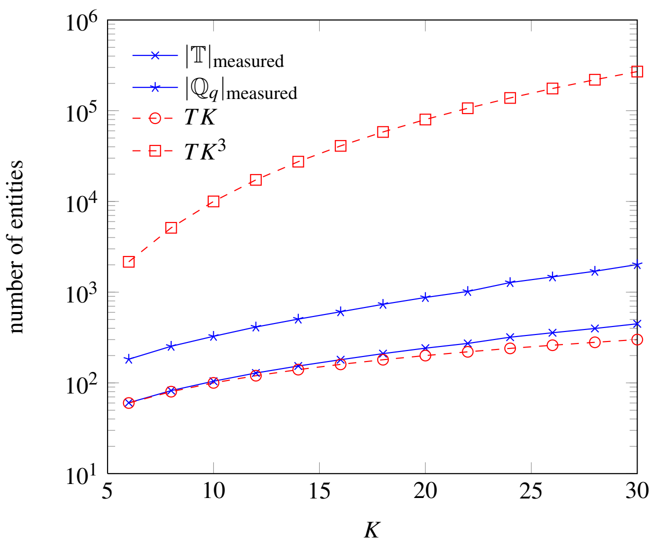


Figure 8*: Verification of the estimated possible combinations for $M = 5$ vertices, with the measured cardinality of the triple set $|\mathbb{T}|_{\text{measured}}$ and the quasi-quadruple set $|\mathbb{Q}_q|_{\text{measured}}$, $K = |\mathbb{W}_n|\ \forall 1 \le n \le N$ and $T = \binom{M}{3}$.*

The simlar shape of both plots, the theoretical complexity and the measured runtime, shows us that our algorithm scales overall with $C_{\text{new}} = \mathcal{O}(TK^3)$. This is reasonable, because once all K^3 combinations per topological triple are checked, the remaining computations are just comparisons of edges. Thus most of the computation is due to the synthesis of consistent triples.

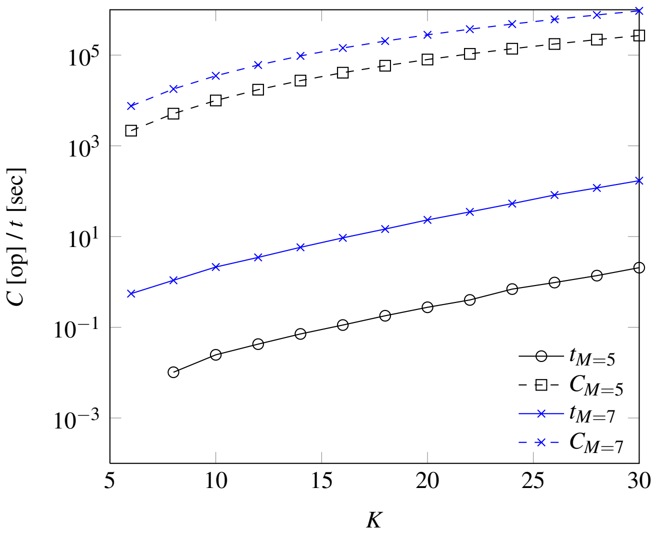


Figure 9*: Validation of the theoretical complexity C and measured runtimes t of the new algorithm for $M = 5$ and $M = 7$ vertices.*

5.4 Comparison to DFS-BT

We compared the runtime of this new approach to the runtime of the algorithm DFS-BT of [3], which counts to the family of Top-Down algorithms. It is shown in [3] that DFS-BT scales strongly with $\mathcal{O}(K^M)$.

In Fig. 10 we plotted the runtimes t for both algorithms over K. It is interesting to see, that for $M = 7$ the influence of K on the runtime is bigger for DFS-BT than for our new algorithm. For $M = 5$ it is not and the new algorithm performs worse.

Another strong advantage of the new approach is that all intermediate solutions represent a consistent solution themselves. This is not the case for DFS-BT so far, where we can only find consistent solutions for the complete graph due to its topology scan.

The brute-force approach would scale with $C_{\text{BF}} = \mathcal{O}(K^N)$ for $N = \binom{M}{2}$. Compared to this both algorithms show a big improvement as they scale with $C_{\text{DFS-BT}} = \mathcal{O}(K^M)$ and $C_{\text{DATEMM,new}} = \mathcal{O}(TK^3)$.

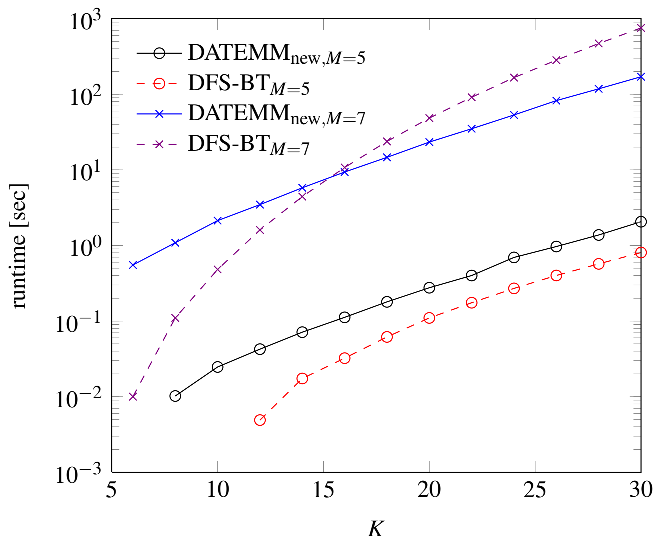


Figure 10*: Comparison of the runtime of the new approach and the synthesis algorithm DFS-BT of [3]*

6. CONCLUSION

We presented an extended algorithm based on DATEMM to find all consistent graphs in a given data set $G(\mathbb{V}, \mathbb{E}, \mathbb{W})$. This new algorithm solves the shortcoming of DATEMM to miss solutions, while it exploits the Bottom-Up strategy. Compared to another synthesis algorithm of the Top-Down family, our new algorithm performs better, when the weights per edge increase.

REFERENCES

[1] Jan Scheuing and Bin Yang, “Disambiguation of TDOA estimation for multiple sources in reverberant environments,” *Transactions on Audio, Speech and Language Processing*, vol. 16, no. 8, pp. 1479–1489, Nov. 2008.

[2] C.M. Zannini, A. Cirillo, R. Parisi, and A. Uncini, “Improved TDOA disambiguation techniques for sound source localization in reverberant environments,” in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE InternationalSymposium on*, 2010, pp. 2666 –2669.

[3] Martin Kreißig and Bin Yang, “Efficient synthesis of consistent graphs,” in *2010 European Signal Processing Conference (EUSIPCO)*, 2010, pp. 1364–1368.

[4] Martin Kreißig and Bin Yang, “A graph theoretical framework for consistent time differences of arrival,” in *ITG Fachtagung Speech Communication 2010*, 2010.

[5] Stuart Russell and Peter Norvig, *Künstliche Intelligenz. Ein moderner Ansatz*, vol. 2, Pearson Studium, 2004.