Synthesis of consistent graphs (SONG)

martin.kreissig@iss.uni-stuttgart.de

December 2, 2013

The document at hand describes in a few summarized steps the attached software for the Synthesis of consistent graphs (SONG) algorithm. For further details refer to the bibliography or source code.

1 Usage

If there is no precompiled mex-file for your Matlab distribution you have to run `song_make` in your Matlab command window, first.

Then the function `song` gives access to the SONG algorithm and requires three input arguments: First input argument is a $2 \times N$ matrix E of natural numbers indicating the start and end vertices of the n-th edge in the graph ($n = 1, \ldots, N$). The second input argument is an $1 \times N$-cell array W that contains the edge weights corresponding to E. Third input argument is a structure that specifies additional parameters for the synthesis:

- `.eps`: is a double value that specifies the threshold for approximate consistency. If zero, exact consistency is applied as the cyclic sum must be exactly zero.
- `.consistency`: is a string that can be choses between full consistency [2] (=*full*) or partial consistency [3] (=*partial*).
- optional part (default values are marked by underline):
  - `.n0`={0,1}: chooses the root vertex of the spanning tree that has either maximal degree (=0) or minimal degree (=1)
  - `.st`={0,1}: defines the type of search for the spanning tree used for determining the fundamental loops (BFS=0, DFS=1).

If the third input argument is missing the default values are `eps`=0, `consistency`=partial, `st`=0 and `n0`=0.

The return argument is a structure array `C` where each element represents a component with its consistent solutions. The components are biconnected (sub)graphs which means that each edge is part of at least one loop and can be checked for consistency. The elements of `C` contain the following entries:

- `.E`: edge matrix of the component (the sorting of the edges may differ from input)
- `.G`: matrix of double values that contains the consistent graphs in each row. The edge weights are sorted according to `E` in the previous bullet. Missing values – which occur in partial consistency – are denoted by `NaN`.

- `.eval`: additional structure with the following values (for evaluation purposes)
  - `.FL_matrix`: represents the fundamental loop (FL) matrix that is used in the synthesis and has dimension $(N - M + 1) \times N$; M is the number of vertices. It can be used to adjust the `eps` value.
  - `.num_of_cFLs_per_FL`: number of consistent weight combinations for each FL, according to the order in the FL matrix previously.
  - `.num_of_cFLs_per_graph`: number of consistent fundamental loops (cFL)s used to synthesize each partially consistent graph. For fully consistent graphs this is equal to the number of FLs. The order corresponds to the consistent graphs in `G`.

2 Theory

Given a graph $G(V, E)$ with vertex set $V = \{v_1, \ldots, v_M\}$ and edge set $E = \{e_1, \ldots, e_N\}$, where each edge e_n has a weight set W_n. A graph $G(V, E, \underline{w})$ with the weight assignment $\underline{w} \in W_1 \times \cdots \times W_N$ is called consistent, when the sums of all loops are zero, e.g. $w_{ij} + w_{jk} + w_{ki} = 0$ for the vertex triple (i, j, k).[1]

Instead of testing all $\prod_{n=1}^{N} |W_n|$ combinations on all possible loops in the graph, the SONG algorithm finds efficiently all consistent weight assignments based on the FLs. Fig. 1 gives an overview of the SONG algorithm.

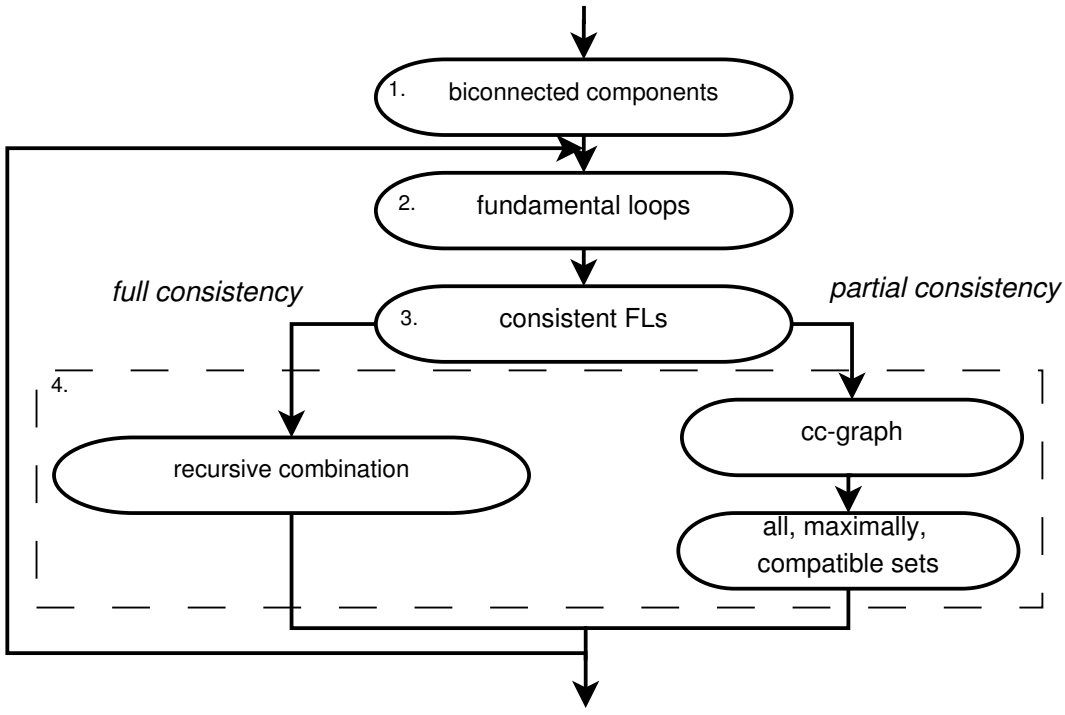

Figure 1: Block diagram

1. Split the given graph topologically in biconnected subgraphs (components). Generally, the input graph can be unconnected and may contain bridge-edges (e_4 in Fig. 2) that must be removed. Moreover, cut vertices (v_3 in Fig. 2) whose removal split a connected graph are

[1] This is the case for weights that stem from difference measures $w_{ij} = p_i - p_j$ like Time Difference of Arrival (TDOA) based speaker localization.

duplicated for each neighboured subgraph. These subgraphs have independent loop sets. This first block finds all biconnected subgraphs by the algorithm of Hopcroft and Tarjan [1] and if necessary splits them. Then each component is processed separately.

2. Compute the FL matrix for each component. Therefore, consider the type of spanning tree (BFS or DFS) and root vertex for the spanning tree (minimal or maximal degree) optionally defined in the 3rd input argument.

3. For each FL $\underline{l}_i \in \{-1, 0, 1\}^N$, all weight combinations are considered to determine its consistent combinations (cFLs). Therein, the approximate consistency is given as $|\underline{l}_i^T \underline{w}| \leq \epsilon \cdot \sqrt{||\underline{l}_i||_0}$ with the ϵ value given in the input (`eps`) and the l_0-norm $||\cdot||_0$ counting the nonzero elements of a vector. The cFLs are stored for the subsequent merging and the number of cFLs for each FLs are stored in `num_of_cFLs_per_FL`.

4. Depending on the chosen consistency, the algorithm of [2] for fully consistent graphs or the algorithm of [3] for partially consistent graphs is chosen.[2] The number of used cFLs are stored in `num_of_cFLs_per_graph`. When the synthesis of one component has finished the next one is being processed.

3 Example

Given the graph in Fig. 2, we illustrate the SONG algorithm.[3]

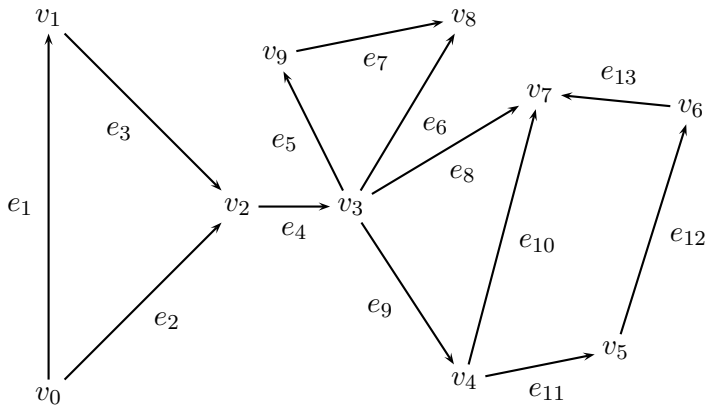

Figure 2: Graph of $M = 10$ vertices and $N = 13$ edges.

The derived edge matrix is

$$E_{\text{in}} = \begin{bmatrix} 0 & 0 & 1 & 2 & 3 & 3 & 9 & 3 & 3 & 4 & 4 & 5 & 6 \\ 1 & 2 & 2 & 3 & 9 & 8 & 8 & 7 & 4 & 7 & 5 & 6 & 7 \end{bmatrix}. \tag{1}$$

The corresponding weight cell array is given in Table 1.

First the graph is splitted in three components, namely $C_1 = \{v_3, v_4, v_5, v_6, v_7\}$, $C_2 = \{v_3, v_8, v_9\}$ and $C_3 = \{v_0, v_1, v_2\}$. Edge e_4 is not contained in any loop and can be removed. Moreover, v_3 represents a cut vertex and thus the incident components C_1 and C_2 are disconnected and both keep v_3.

Then for each component the FLs are determined and the sets of cFLs are computed. The cFLs are combined either to fully consistent graphs according to [2] or to partially consistent graphs

[2] Even though the synthesis of partially consistent graphs produces fully consistent graphs, too, the algorithm in [2] is more efficient for the direct computation.

[3] This example is given in the attached Matlab demo.

W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}	W_{11}	W_{12}	W_{13}
0.98	1.11	-0.09	1.39	12.00	6.15	-5.96	8.11	-1.11	4.00	2.15	4.85	1.89
2.06	-0.89	-1.99			-8.08	2.9	-0.89	-0.11	9.06	1.01	0.93	
		-0.62							-0.89			

Table 1: Possible weights for the graph of Fig. 2

according to [3]. The result of the ith component is returned as a seperate edge matrix $E_{\text{out},i}$ indicating the contained vertices and its consistent weight matrix G_i. For partial consistency the missing edge weights of a consistent subgraph are given by the value NaN. With $\epsilon = 0.15$ SONG returns the solutions shown in Table 2.

$E_{\text{out},1}$:	3	4	4	5	3	6
	4	5	7	6	7	7
G_1 :	-1.11	2.15	9.06	4.85	8.11	1.89
	-0.11	NaN	-0.89	NaN	-0.89	NaN
	NaN	1.01	4.00	0.93	NaN	1.89

$E_{\text{out},2}$:	3	3	9
	8	9	8
G_2 :	6.15	12.00	-5.96

$E_{\text{out},3}$:	0	1	0
	2	2	1
G_3 :	1.11	-0.09	0.98
	-0.89	-1.99	0.98

Table 2: All approximately, partially consistent graphs from Fig. 2 with weights of Table 1

References

[1] John Hopcroft and Robert Tarjan. Efficient Algorithms for Graph Manipulation. *Communications of the ACM*, 16:372–378, 1973.

[2] Kreißig, Martin and Yang, Bin. An efficient algorithm for the synthesis of fully consistent graphs. *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, 2012.

[3] Martin Kreißig and Bin Yang. Reliable Simultaneous TDOA Assignment in Multi-Source Reverberant Environments. *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, 2013, (submitted paper).