A computational implementation of Vector-based 3D Graphic Statics (VGS) for interactive and real-time structural design

Jean-Philippe JASIENSKI[a], Yuchi SHEN[b], Patrick Ole OHLBROCK[c], Denis ZASTAVNI[a], Pierluigi D'ACUNTO[cd]

[a] UCLouvain, LOCI – Structures & Technologies
[b] Southeast University, School of Architecture
[c] Technical University of Munich, TUM School of Engineering and Design, Professorship of Structural Design
[d] Technical University of Munich, Institute for Advanced Study

---

Article Info

Keywords:
computational structural design
graphic statics
form diagram
force diagram; planar graph
planarization
parallel transformations
constraint-driven transformations.

Abstract

This article presents a computational implementation for the Vector-based Graphic Statics (VGS) framework making it an effective CAD tool for the design of spatial structures in static equilibrium (VGS-tool). The paper introduces several key features that convert a purely theoretical graph and geometry based framework into a fully automated computational procedure, including the following new contributions: a general algorithm for constructing 3-dimensional interdependent force and force diagrams; the implementation of a procedure that allows the interdependent transformation of both diagrams; an approach to apply specific constraints to the computationally generated diagrams; the integration of the algorithms as a plug-in for a CAD environment (Grasshopper3D of Rhino3D). The main features of the proposed framework are highlighted with a design case study developed using the newly introduced CAD plug-in (namely the VGS-tool). This plugin uses synthetic-oriented and intuitive graphical representation to allow the user to design spatial structures in equilibrium as three-dimensional trusses. The goal is to facilitate collaboration between structural engineers and architects during the conceptual phase of the design process.

---

1. Introduction

1.1 Graphic statics and structural design

Graphic statics provides intuitive methods to design efficient and elegant structures. It involves the use of form and force diagrams, with the former representing the geometry of a structure in static equilibrium and the loads acting on it and the latter representing the equilibrium of forces for each node of the structure (*Rankine, 1858; Maxwell, 1864; Culmann, 1866; Cremona, 1872*). The graphical nature of the two diagrams offers a visual and intuitive understanding of the relationship between form and forces in a structure, which facilitates the structural design process (*Zalewski and Allen, 1998*). Swiss engineer Robert Maillart, amongst others, used graphic statics to design new structural forms such as the Chiasso shed in 1924 (*Zastavni, 2008*), the Salginatobel Bridge in 1929 *(Fivet and Zastavni, 2012*), and the Vessy Bridge in 1936 (*Zastavni et al., 2014*). Moreover, contemporary structural engineers such as Jurg Conzett, Joseph Schwartz and Bill Baker from Skidmore Owings & Merril (*Beghini et al., 2014*) have utilized this approach in their work. In recent years, comprehensive research has been conducted to extend graphic statics to the third dimension (*Jasienski et al., 2014*). In this context, two formulations of the problem have been mainly pursued: the *polyhedron-based* (*Konstantatou et al., 2018; Akbarzadeh, 2016; Lee, J., 2019*) and the *vector-based* (*D'Acunto et al., 2019*) approaches. One of the main reason to pursue the development of vector-based graphic statics in 3D is that the graphical forms the human perceives more accurately are points and linear elements, including their position, lengths and angles (*Mackinlay, 1986*).

1.2 Problem statement and objectives

Implementing 3D graphic statics within a computational environment has the potential to provide an invaluable resource for the design of spatial structures in static equilibrium. Such projects are under development for the polyhedron-based approach, including Polyframe (*Nejur and Akbarzadeh, 2021*), compas_3GS (*Lee et al., 2018*) and 3DGS (*Milošević and Graovac, 2023*).

Within the domain of vector-based graphic statics, the algebraic graph approach (*Van Mele & Block, 2014; Alic and Åkesson, 2017*) was computationally implemented but it only addresses the case of 2D structures whose form diagrams have underlying planar graphs, i.e., graphs that can be drawn on the plane without edge intersections.

In the 3D case, a vector-based force diagram can be readily assembled by manual constructions using iterative simple geometric operations within a 3D software environment for a given form diagram (*Jasienski et al., 2016*; *D'Acunto et al., 2019*). However, this procedure requires the user to have a specific knowledge and is very time-consuming for complex structures. An even minor modification of the initial setup – such as changes in the topology of the structure, applied loads or position of supports – almost always implies the entire new reconstruction of the diagrams. This shortcoming renders the manual approach inconvenient for the design of complex 3D structures, especially in the conceptual design phase when several design variations are usually tested. Some unpublished partial computational workflow existed but were case-specific and not fully automated.

1.3. Contribution

This paper introduces a new computational framework for the automated construction of vector-based interdependent form and force diagrams for any 2D and 3D pin-jointed truss structures with planar or non-planar underlying graphs. Two alternative algorithms are developed (namely the MED and the QUAD) for the assembly of the force diagram. In the non-planar case, each algorithm corresponds to a different strategy for the automated planarization of the underlying graph of the form diagram, thus leading to different configurations of the force diagram since there is no unique way to planarize a non-planar graph.

The paper also presents the implementation of these algorithms into a grasshopper3D plugin. Some new features such as the form finding of new structures at equilibrium are presented for the first time.

1.4 Content

This article is organized as follows. Section 2 briefly highlights the key features of the theoretical background upon which the presented computational implementation is based. Section 3 describes the computational process, from the general scheme to the core steps of the procedure. Section 4 represents the main contribution of this research and describes the algorithm that planarizes non-planar graphs, which is necessary to construct the force diagrams. Section 5 presents the integration of the computational procedure as a plugin in the CAD environment of Grasshopper3D in Rhino3D (*McNeel, 2023*). Finally, Section 6 illustrates the potential of the proposed computational framework for structural design with a case-study.

1.5 Notation

For a given structure in static equilibrium, three classes are used in the proposed computational framework to represent the structure's form (***F***), force (***F****) and topological (***T***) diagrams, the latter corresponding to the underlying graph of ***F***. $\boldsymbol{T_P}$ refers to a planar embedding (i.e. plane graph) of ***T***; if ***T*** is non-planar, it is first planarized into a planar graph through a computational routine. The index ***c*** denotes a graph generated in the computational environment. The index ***i*** designates an intermediary version of the graph ($\boldsymbol{T_i}$) that is modified during an iterative loop. $\boldsymbol{T_{mc}}$ and $\boldsymbol{T_{rc}}$ are the result of splitting the graph $\boldsymbol{T_c}$ in two graphs, one being the maximum planar graph, the other being the graph containing the remaining edges. The notation used to describe the constituting elements of the three classes ***T***, ***F*** and ***F**** is presented in Table 1, as well as their structs and attributes, constituting the data structure of the algorithms presented in this contribution. The graphical convention for tension, compression and external forces is illustrated in Fig. 1.

Table 1: Algorithmic Data structure & notation.

Class	***Structs***		***Attributes***		
Form diagram ***F***	Vertices	V_i	ID		$i \dots j$
			Coordinate		$V_i \vert [X, Y, Z]$
	Edges	E_{i-j}^k	Adjacency		$[i - j]$
			Duplicate Identity		k
			Type	Inner Force	$-1/1$
				External Force	0
			Force magnitude		f
			Correspondence		$\{[q, p, k] \dots\}$
Force diagram ***F****	Vertices	V_q	ID		$q \dots p$
			Coordinate		$V_q \vert [X, Y, Z]$
	Edges	$E_{q-p}^{k\;*}$	Adjacency		$[q - p]$
			Duplicate Identity		k
			Type	Inner Force	$-1/1$
				External Force	0
			Edge Length		$\vert f \vert * scale$
			Correspondence		$[i, j, k]$
	Cycles of force vectors	$\langle V_i \rangle^*$	ID		$i \dots j$
Topological diagram ***T***	Cycles	$\langle V_i \rangle$	Embed Edge Order		$v_i \vert \{E_{i-j}^k \dots\}$
	Cycle of auxiliary vertices	$\langle n_i \rangle$	Embed Edge Order		$n_i \vert \{E_{i-j}^k \dots\}$
	Assemble Sequence	$\{v_{i,} \dots\}$	Cycle ID		$\langle v_i \rangle$ or $\langle n_i \rangle$
			Related Edge		$v_i\ to\ v_j \vert E_{i-j}^k$

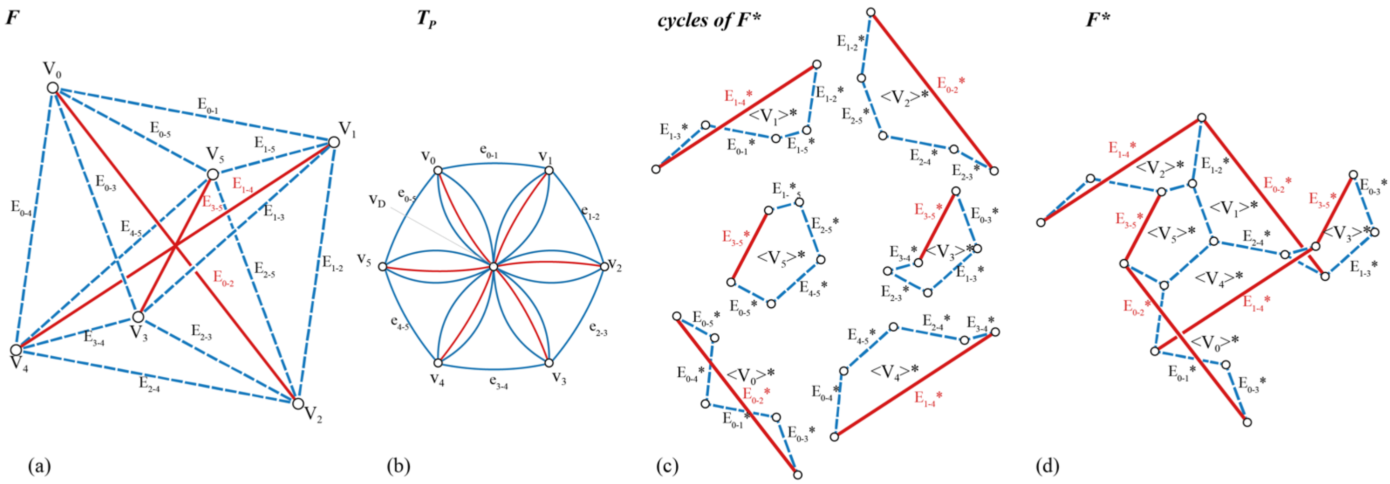


Fig. 1: (a) 3D form diagram ***F*** of a self-stressed octahedron; (b) Plane graph ***Tp*** (in colours); (c) individual closed cycles of force vectors representing the static equilibrium of each node in the self-stressed tetrahedron. (d) 3D force diagram ***F****. Structural members that are in compression are in blue, those in tension are in red.

2. Theoretical background

2.1 Vector-based graphic statics

The computational framework presented in this paper is based on the vector-based graphic statics approach, which was initially introduced by Maxwell (1864). In this approach, the equilibrium of the forces acting on a node V_i of ***F*** is represented by a close cycle of force vectors $<V_i>^*$ in ***F****. Moreover, for each pair of opposite forces acting within the same edge $E_{i\text{-}j}$ of ***F***, two opposite force vectors exist in ***F****, each belonging to distinct closed cycles of force vectors $<V_i>^*$ and $<V_j>^*$. When two such opposite force vectors overlap in ***F****, a force edge $E_{i\text{-}j}^*$ replaces them *(D'Acunto et al., 2019*). The diagrams are reciprocal in the special case that ***F*** and ***F**** have an equal number of edges (*Crapo, 1979*). Otherwise, non-overlapping force vectors exist (*Jasienski et al., 2016*), and the diagrams are not reciprocal (Fig. 2)

2.2 Assembly of the force diagrams: a graph theory-based approach

The general approach to constructing ***F**** is to derive the underlying graph ***T*** of ***F*** and use its planar embedding $\boldsymbol{T_p}$ and its corresponding dual graph as a reference for generating ***F****. Depending on how ***T*** is planarized into $\boldsymbol{T_p}$ (*Tarjan, 1970; Beneke and Pippert , 1978; Brandes, 2000; Buchleim et al., 2013*), different configurations of ***F**** are available, each characterized by a specific organization of the cycles of force vectors within the diagram (*D'Acunto et al. 2019*). A possible way to manually generate a plane graph $\boldsymbol{T_p}$ of ***T*** is to successively split its crossing edges and reconnect them to one or more newly introduced *auxiliary vertices* v_{Di} while fulfilling the static equilibrium of every node of the structure (*D'Acunto et al. 2019*).

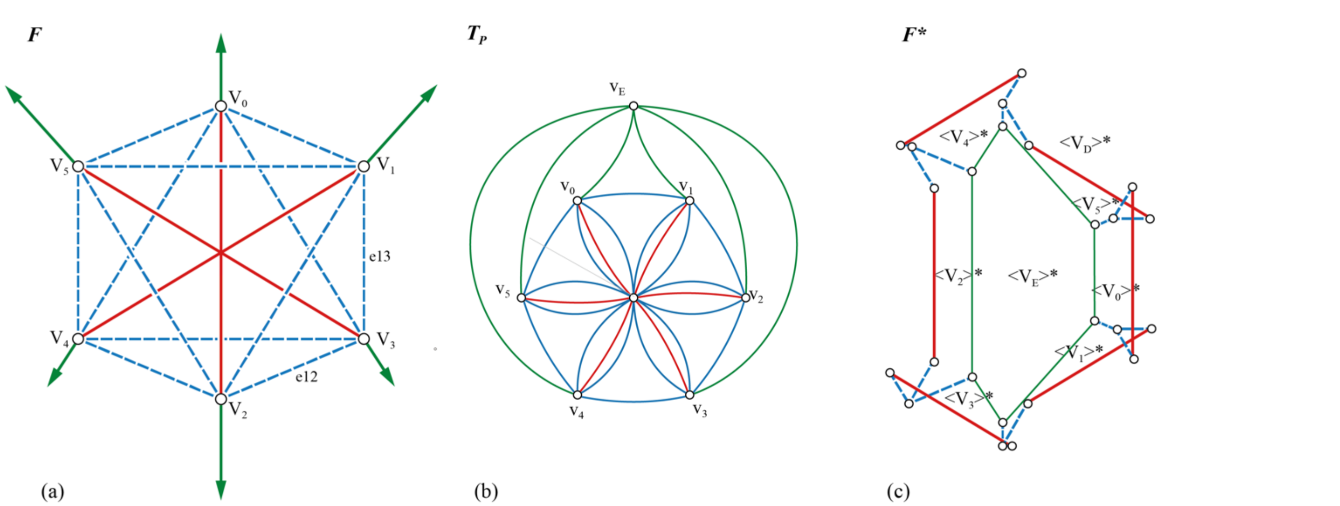


Fig. 2: Externally loaded octahedron: (a) Form Diagram ***F*** (b) plane graph ***Tp***; (c) 3D force diagram ***F****. F and F* are not reciprocal diagrams because the initial graph corresponding to the structure ***T*** is not planar. The non-overlapping vectors can be identified as those vectors represented twice in (c).

3 Computational implementation

3.1 Overview of the computational setup

This section outlines the full computational implementation of the theoretical framework briefly introduced in Section 2, namely the VGS algorithm. A general scheme is presented in Fig. 3, and the algorithm's main steps are described in section 3. The tool's main function is to generate automatedly interdependent form and force diagrams for a given arbitrary 2D or 3D structure with applied forces in static equilibrium.

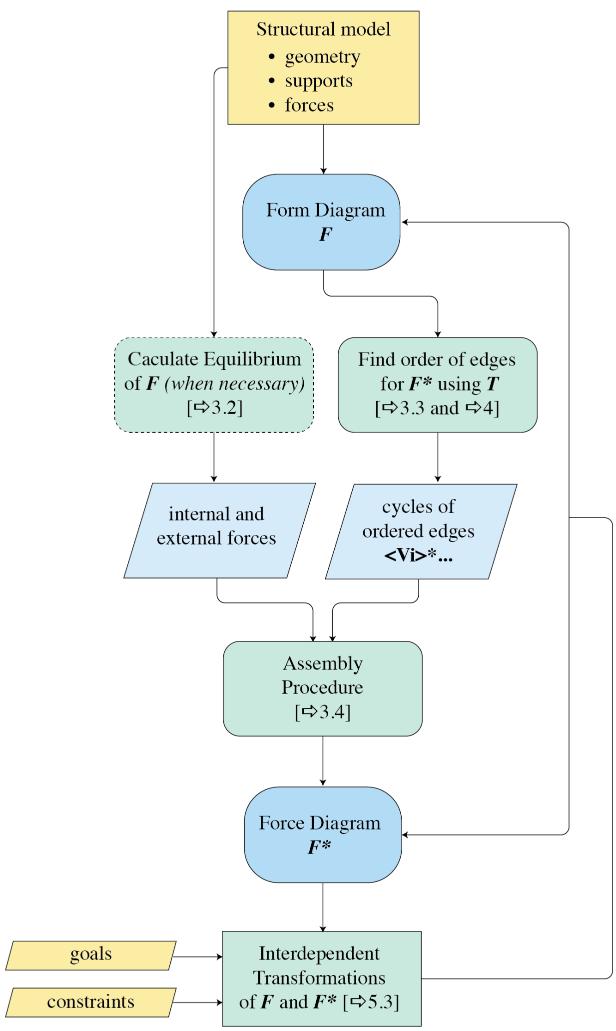


Fig. 3: Overview of the general algorithmic procedure (VGS algorithm) that automatedly generates interdependent form and force diagram from a given structural model.

The preliminary step is to provide a geometry, supports and forces (external or internal) that together compose a valid discrete structural model.

The first step of the algorithm is to generate a form diagram ***F*** based on the input geometry, supports and forces. If the structural model provided in the initial setup is not in static equilibrium, a numerical solver will calculate the magnitude of the internal and external forces (see section 3.2).

The second step involves finding an ordered sequence for the edges of each vertex that will compose the force diagram (see section 3.3). Two alternative planarization algorithms that are necessary to perform this task are detailed in section 4.

Thanks to these two sets of data, an assembly procedure generates a force diagram ***F**** corresponding to the previously defined form diagram ***F*** (see section 3.4).

Eventually, the two diagrams are made interdependent from each other through numerical methods. This allows the user to apply specific transformations and constraints to one of the diagrams and assess in real time how it affects the other diagram (see section 5.3).

3.2. Evaluation of the equilibrium

The equilibrium of the structure's nodes and the calculation of the magnitude of the internal forces are, by default, solved geometrically node-by-node or numerically after setting up the equilibrium matrix of the structure (*D'Acunto et al., 2019*). However, this initial information could also be provided by other equilibrium solvers and form-finding tools. It should also be noted that solving the equilibrium problem is not a strict prerequisite to assembling form and force diagrams. On the contrary, the VGS-tool introduces the possibility to enforce the static equilibrium of an arbitrary structure by relying on the transformation function (see Section 5.3).

*3.3 Finding the order of force vectors in the force cycles constituting F**

For a given ***F***, this part of the algorithm provides the specific order of force vectors used to construct the closed cycles of forces vectors *<V_i>** constituting ***F****. The algorithm uses the Boyer-Myrvold script for planarity testing (*Boyer and Myrvold, 2004*). First, the graph $\boldsymbol{T_c}$ is generated from ***T***. To this end, all the edges and vertices of ***F*** are identified and stored in a list of lists composed of a list of the vertices and after one list per vertex containing all its edges. When external forces (i.e. applied forces and support forces) exist, a new vertex $\mathbf{V_E}$ is created, and new edges connecting $\mathbf{V_E}$ to the nodes (*Jasienski et al. 2016*) where the external forces are applied are added to the list of edges. After that, the planarity check algorithm is performed on $\boldsymbol{T_c}$. If $\boldsymbol{T_c}$ is not planar, a specific planarization algorithm (based on the choice made by the user – see Section 4) is implemented to modify the graph $\boldsymbol{T_c}$ iteratively ($\boldsymbol{T_{ic}}$) until it is converted into a planar graph $\boldsymbol{T_{pc}}$. From $\boldsymbol{T_{pc}}$, a list of clockwise-ordered edges can be extracted for each node of the structure. Retrieving this information is equivalent to defining the dual graph of $\boldsymbol{T_{pc}}$, which corresponds to the underlying graph of ***F****. The lists of clockwise-ordered edges are subsequently used by an algorithm to assemble the cycles of force vectors and, eventually, the entire ***F**** (see Section 3.4.).

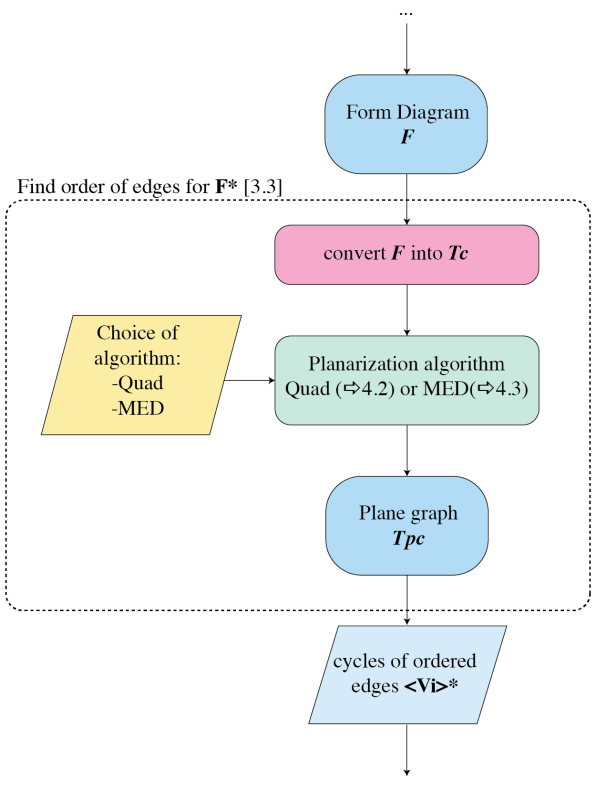


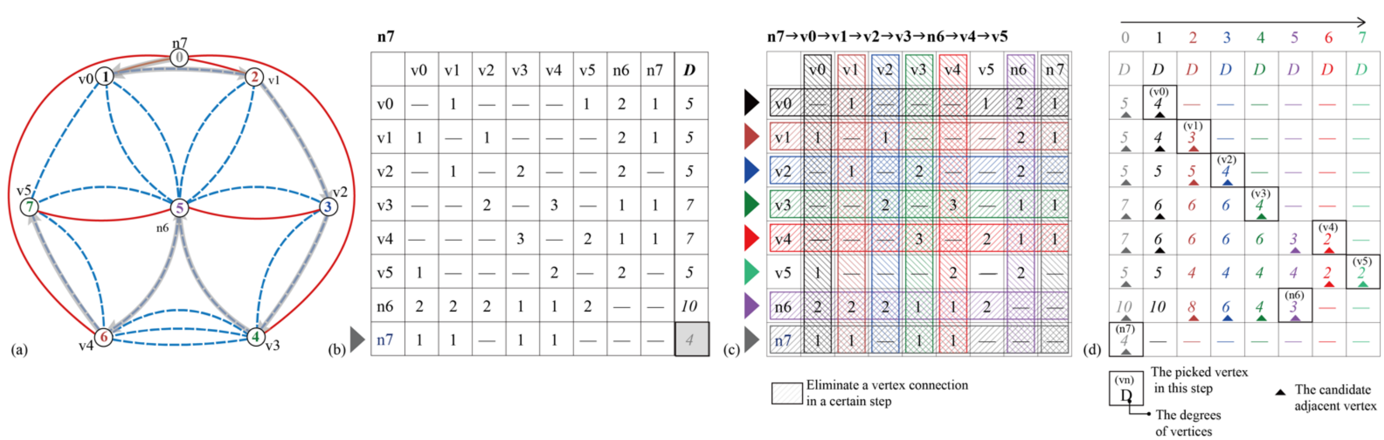
Fig. 4: Algorithm to find the order for edges of ***F**** *(see section 3.3)*.

3.4. Assembly procedure

After all the cycles $<V_i>^*$ of ***F**** have been determined, a specific sequence for assembling them is required to construct a complete ***F****. This sequence is composed of vertices of ***F*** Seq = {...,*Vk*,*Vj*,...} fulfilling the condition that the two adjacent cycles of force vectors *<Vk>* <Vj>** share at least one edge, which is made of two opposite force vectors that refer to the same edge of ***F***. This problem could be summarized as such: finding a one-way path that visits once all the vertices of the plane graph $\boldsymbol{T_{pc}}$.

The method developed to solve this problem uses an elimination procedure that is illustrated in Fig. 5. The algorithm works with the adjacency matrix of the graph $\boldsymbol{T_{pc}}$ and always starts from the least connected vertex (i.e. the vertex that counts the least row element number in the adjacency matrix). The procedure repeats as follows. After a vertex v_n is added to the sequence, for all the related vertices that are connected to v_n the adjacent element in their rows is removed. The next vertex is selected in the rows of v_n, fulfilling the criteria of having the least count of the row elements of the vertex connecting to v_n.

In some cases, a one-way path that connects all the vertices of a graph cannot be established due to its topology, resulting in a gap in the assembly sequence. This happens if the selected vertex has no row element because of the previous elimination process. In this case the algorithm starts from another vertex that is connected to a vertex which was selected in the previous iteration. The elimination procedure then carries on until all the vertices of the graph have been selected.



	v0	v1	v2	v3	v4	v5	n6	n7	*D*
v0	—	1	—	—	—	1	2	1	5
v1	1	—	1	—	—	—	2	1	5
v2	—	1	—	2	—	—	2	—	5
v3	—	—	2	—	3	—	1	1	7
v4	—	—	—	3	—	2	1	1	7
v5	1	—	—	—	2	—	2	—	5
n6	2	2	2	1	1	2	—	—	10
n7	1	1	—	1	1	—	—	—	4

Fig. 5: Assembly procedure. (a) ***Tp*** with the sequence of the assembly procedure represented by a grey arrow, with the order corresponding to the number at each vertex. The adjacency matrix is represented on the right (b, c, d) with the elimination procedure in colour.

4. Algorithm for the planarization of the topological graph

The present section describes the algorithm for the automated planarization of the topological graph of the structure. Two variations of the algorithim are presented, namely the *QUAD* (Section 4.2) and the *MED* (Section 4.3). For both, the presented approach relies on an incremental process that starts from a reduced planar subgraph (the maximal planar graph – see Section 4.1.1) that is successively enlarged to correspond to a suitable $\boldsymbol{T_{pc}}$ (meaning that this planar embedding contains all the initial edges of $\boldsymbol{T}$). Hence the algorithmic implementation processes in an opposite manner compared to the theoretical procedure (resumed in Section 2). Irrespective of the specific procedure that is used, the result is a suitable planar graph $\boldsymbol{T_{pc}}$ that contains all the required information to assemble the force diagram ***F****. Each algorithm leads to a different type of configuration of force diagram that can prove more visually adequate for different structural typologies. The definition of this algorithm is key to the present contribution since it determines the resulting configuration of ***F****.

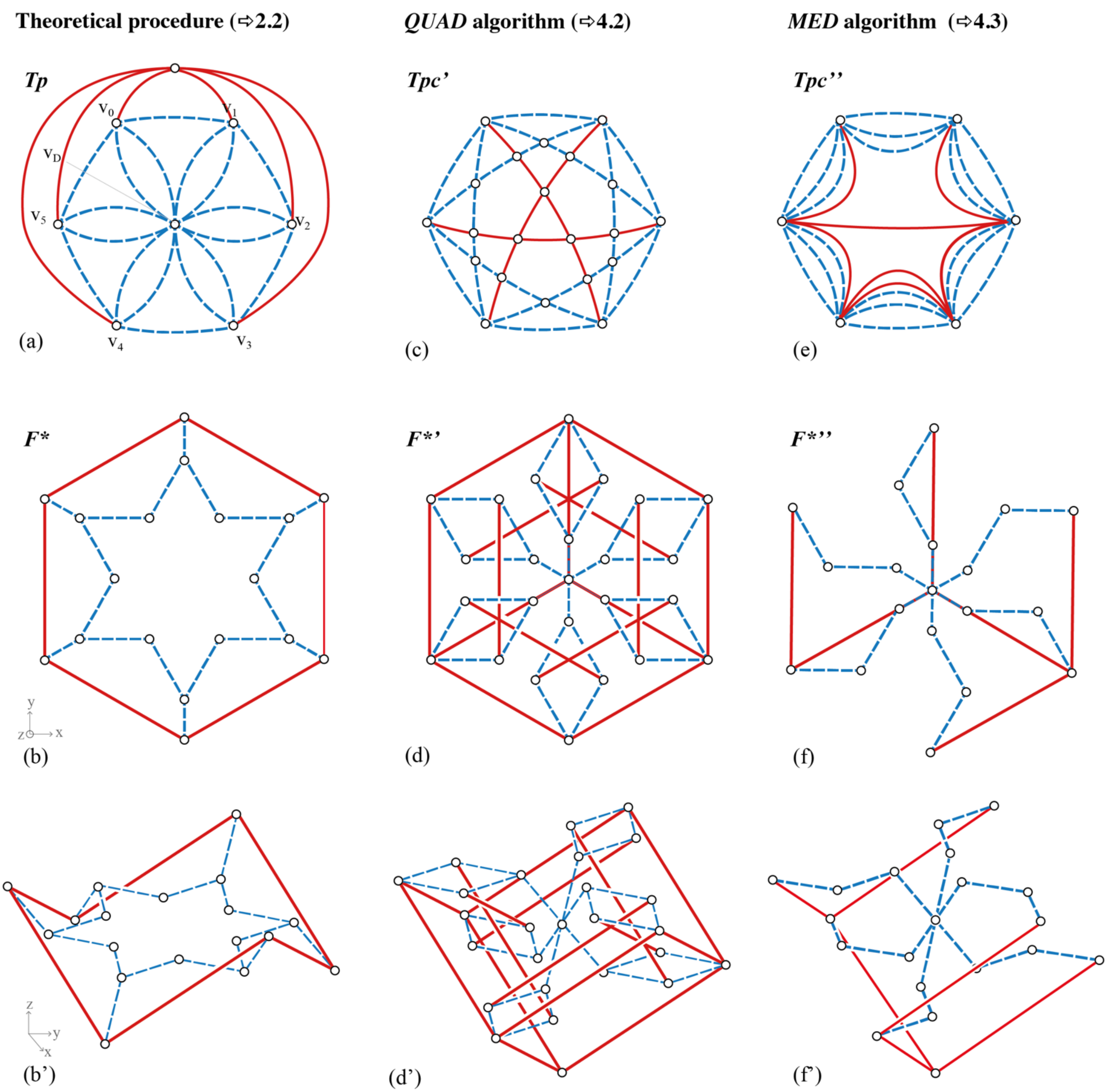


Fig. 6. Comparison of the planarized topological graph and the corresponding form diagram for the Force diagram ***F*** presented in Figure 1 (a)(c)(e): general procedure (***Tp*** (a) and ***F* (b) and (b')***), QUAD algorithm procedure (***Tpc' (c)*** and ***F*'(d) and (d')'***), MED algorithm procedure (***Tpc''(e)'*** and ***F*'' (f) and (f')***).The two last lines of figures represent different views of the resulting force diagram (middle line is top view, bottom line is side view)

4.1. Preliminary step – finding the maximal planar graph

The preliminary step for both *QUAD* and *MED* procedures is to generate the maximal planar graph of $\boldsymbol{T_c}$. Indeed, for any non-planar graph, it is always possible to find a planar graph that is a subgraph of it (*Harary, 1969*). The computationally generated topological graph $\boldsymbol{T_c}$ is consequently split into two graphs: the maximal planar graph $\boldsymbol{T_{mc}}$ and the graph containing all the remaining edges $\boldsymbol{T_{rc}}$ so that:

$$\boldsymbol{T_c} = \boldsymbol{T_{mc}} \cup \mathbf{T_{rc}}.$$

Two different algorithms are mainly used to find the maximum planar graph (*Tamassia, 2013*): namely the *Vertex Increment method* (VIM) and *Edge Increment method* (EIM) (*Jayakumar et al., 1989*). The principles of these two procedures are illustrated in Fig. 7.

Both methods provide a planar graph $\boldsymbol{T_{mc}}$ as a valid solution but cannot ensure that it is the exact maximum planar graph (which is a nondeterministic polynomial-time complete problem). The task has a complexity of $O(n^2)$ at worst case for both VIM and EIM. In the scope of the present research, several experiments were set up to compare their efficiency in the context of the VGS algorithm. Both methods were tested on randomly generated non-planar graphs $\boldsymbol{T} = \boldsymbol{T_p} \cup \boldsymbol{T_r}$, where $\boldsymbol{T_p}$ is a triangulated planar mesh graph, and $\boldsymbol{T_r}$ is the set of edges added to make the graph non-planar (see Fig. 8).

The planarity rate is defined as:

$$\text{Rs} = \boldsymbol{T_r}.\ \boldsymbol{R_s} = \mathbf{T_r}.edges\_count/\boldsymbol{T}.edges\_count.$$

Two statistics studies are carried out on $\boldsymbol{R_s} = 25\%$ and $\boldsymbol{R_s} = 70\%$ on graphs with incremented vertex number (20~100). The results are displayed in Fig. 9. The analysis of the results shows that EIM is more efficient than VIM on planarizing the graphs that have a low $\boldsymbol{R_s}$. VIM works slightly better in the graphs that have a high $\boldsymbol{R_s}$. With all the above results in regard, VGS-tool implements EIM by default to fit the most case for better planarization efficiency and accuracy.

The EIM starts from a planar subgraph $\boldsymbol{T_{imc}}$ which at the first iteration only contains all the adjacent edges of the first vertex $\mathbf{v_0}$ (and in presence of external forces of v_E as well) and adds one vertex at each iteration. At an iteration resulting in $\boldsymbol{T_{imc}}$ being not planar anymore, the algorithm will check all the adjacent edges to the edge that was added and find the most edges that can be added to $\boldsymbol{T_{imc}}$ and keep it still planar. In each iteration, the identified edges will be added to $\boldsymbol{T_{irc}}$ and disregarded for the other vertices. At the end of the algorithm:

$$\boldsymbol{T_{irc}\ (=T_{rc})} \cup \boldsymbol{T_{imc}\ (T_{mc})} = \boldsymbol{T_c}.$$

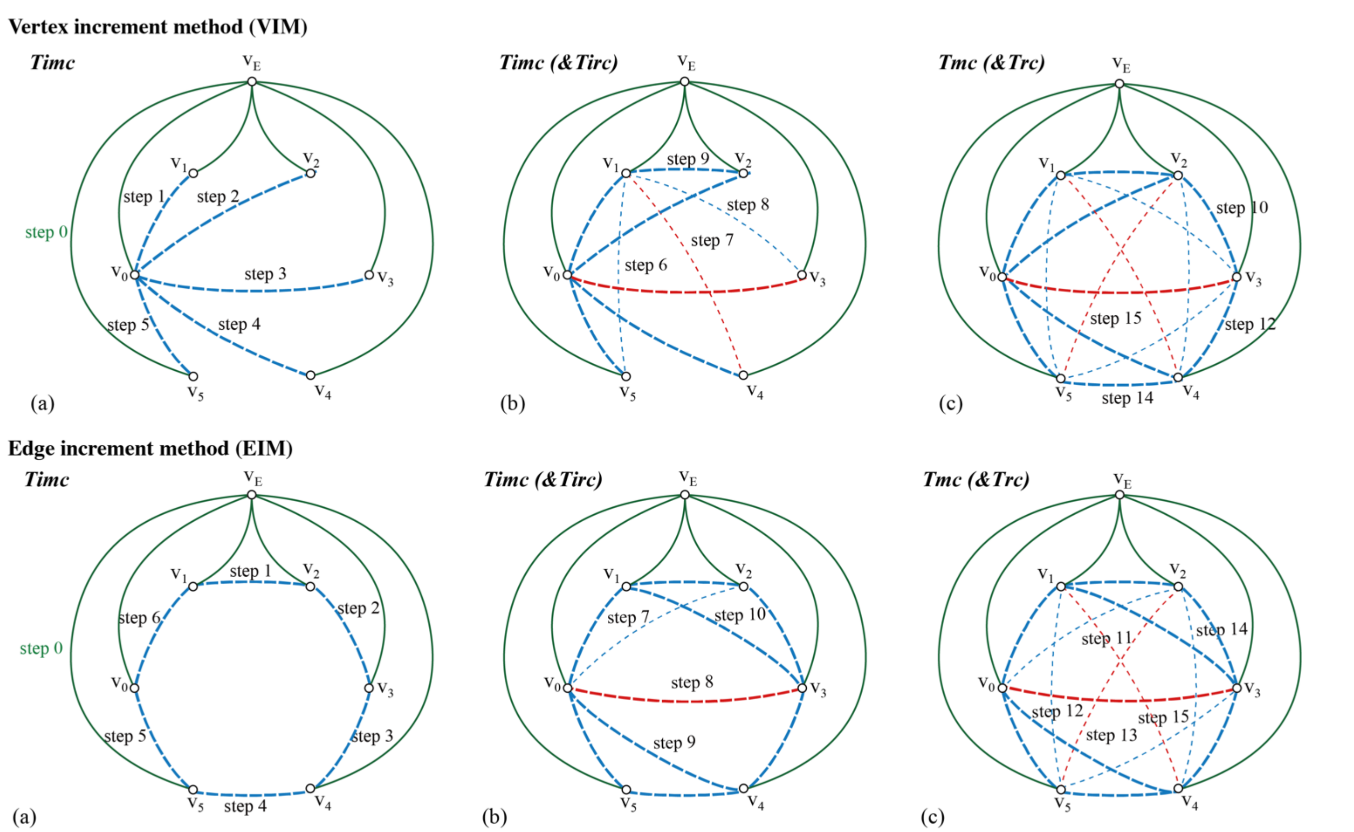


Fig.7 : Principles of Vertex Increment method (top) and Edge increment method (bottom). VIM adds back each edge connected to the same vertex, testing if the graph is still planar after an edge is added. Then the procedure goes to the next vertex. EIM adds back one edge after, , testing if the graph is still planar after an edge is added The intermediate steps are represented in (a) and (b) and the final maximal planar graph ***Tmc*** are represented with the graph of remaining edges $\boldsymbol{T_{rc}}$ in (c).

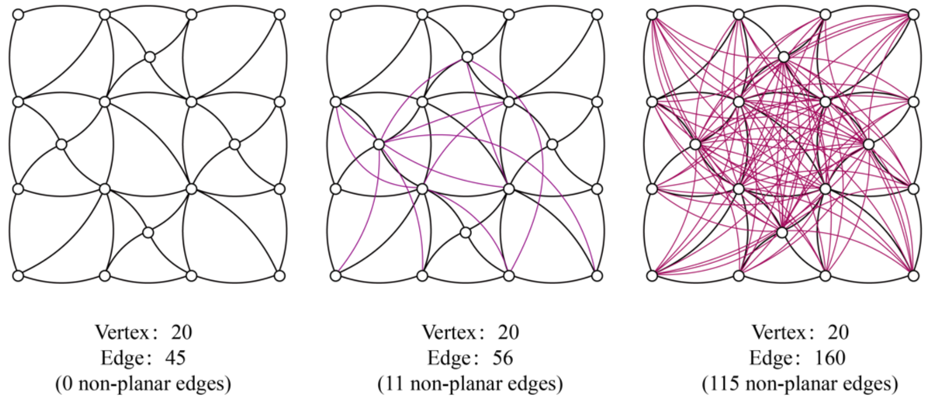
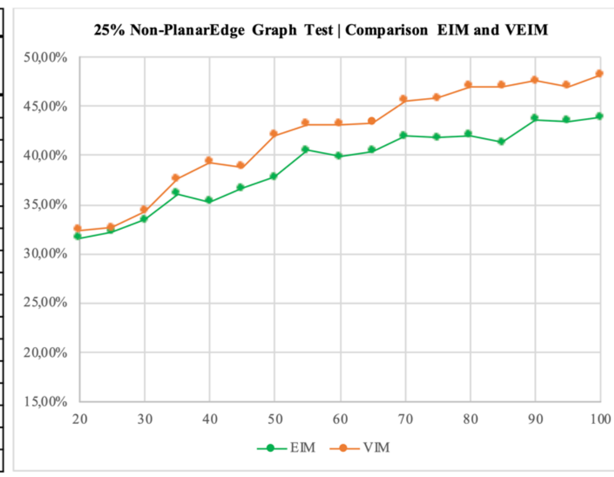


Fig. 8 : Example of graphs used for the accuracy test. (a) The planar triangulated mesh graph Tp, (b) The graph T = Tp U Tr (represented in magenta) with Rs = 25% (c) the graph T = Tp U Tr (represented in magenta) with Rs = 70%. (b) and (c) correspond respectively to the first row of the two tables in Fig 9.

Graph			Edge Increment Method (EIM)			Vertex Increment Method (VIM)			
V(count)	E(count)	Non-Planar Edge Rate	Tpc	Trc	Trc/E(count)	Tpc	Trc	Trc/E(count)	Best Method
20	56	**25%**	38,31	17,69	31,59%	37,87	18,13	32,38%	**E**
25	74	**25%**	50,14	23,87	32,25%	49,86	24,14	32,63%	**E**
30	92	**25%**	61,24	30,76	33,44%	60,38	31,63	34,38%	**E**
35	110	**25%**	70,30	39,70	36,09%	68,58	41,42	37,65%	**E**
40	133	**25%**	86,10	46,90	35,26%	80,70	52,30	39,32%	**E**
45	151	**25%**	95,70	55,30	36,62%	92,40	58,60	38,81%	**E**
50	170	**25%**	105,80	64,20	37,76%	98,50	71,50	42,06%	**E**
55	189	**25%**	112,40	76,60	40,53%	107,50	81,50	43,12%	**E**
60	208	**25%**	125,00	83,00	39,90%	118,20	89,80	43,17%	**E**
65	226	**25%**	134,70	91,30	40,40%	128,10	97,90	43,32%	**E**
70	245	**25%**	142,30	102,70	41,92%	133,40	111,60	45,55%	**E**
75	264	**25%**	153,60	110,40	41,82%	143,00	121,00	45,83%	**E**
80	280	**25%**	162,30	117,70	42,04%	148,40	131,60	47,00%	**E**
85	298	**25%**	175,00	123,00	41,28%	158,00	140,00	46,98%	**E**
90	316	**25%**	178,20	137,80	43,61%	165,80	150,20	47,53%	**E**
95	335	**25%**	189,30	145,70	43,49%	177,40	157,60	47,04%	**E**
100	353	**25%**	198,10	154,90	43,88%	183,00	170,00	48,16%	**E**



Graph			Edge Increment Method (EIM)			Vertex Increment Method (VIM)			
V(count)	E(count)	Non-Planar Edge Rate	Tpc	Trc	Trc/E(count)	Tpc	Trc	Trc/E(count)	Best Method
20	160	**70%**	51,42	108,58	67,86%	51,74	108,26	67,66%	**E**
25	213	**70%**	67,33	145,67	68,39%	66,67	146,33	68,70%	**V**
30	257	**70%**	78,00	179,00	69,65%	78,67	178,33	69,39%	**E**
35	307	**70%**	89,00	218,00	71,01%	94,00	213,00	69,38%	**V**
40	353	**70%**	96,00	257,00	72,80%	105,67	247,33	70,07%	**V**
45	403	**70%**	105,33	297,67	73,86%	114,00	289,00	71,71%	**V**
50	453	**70%**	121,33	331,67	73,22%	123,33	329,67	72,77%	**V**
55	503	**70%**	130,67	372,33	74,02%	135,33	367,67	73,09%	**V**
60	553	**70%**	136,00	417,00	75,41%	145,67	407,33	73,66%	**V**
65	603	**70%**	145,00	458,00	75,95%	156,67	446,33	74,02%	**V**
70	653	**70%**	152,33	500,67	76,67%	168,00	485,00	74,27%	**V**
75	703	**70%**	154,67	548,33	78,00%	177,33	525,67	74,77%	**V**
80	747	**70%**	172,00	575,00	76,97%	185,33	561,67	75,19%	**V**
85	793	**70%**	182,33	610,67	77,01%	192,67	600,33	75,70%	**V**
90	843	**70%**	185,67	657,33	77,98%	202,67	640,33	75,96%	**V**
95	893	**70%**	196,33	696,67	78,01%	211,67	681,33	76,30%	**V**
100	940	**70%**	205,67	734,33	78,12%	224,33	715,67	76,13%	**V**

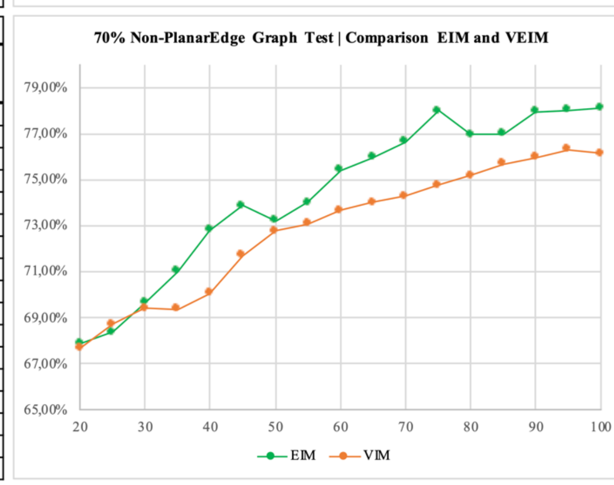


Fig. 9: Results of the experiment comparing the efficiency of EIM and VEIM for 25% non-planar edge graph (top-left) and 70% non-planar edge graph (bottom-left). Graphical results of the experiment comparing the efficiency of EIM and VEIM for 20% non-planar edge graph (top-right) and 70% non-planar edge graph (bottom-right).

4.2 The QUAD procedure

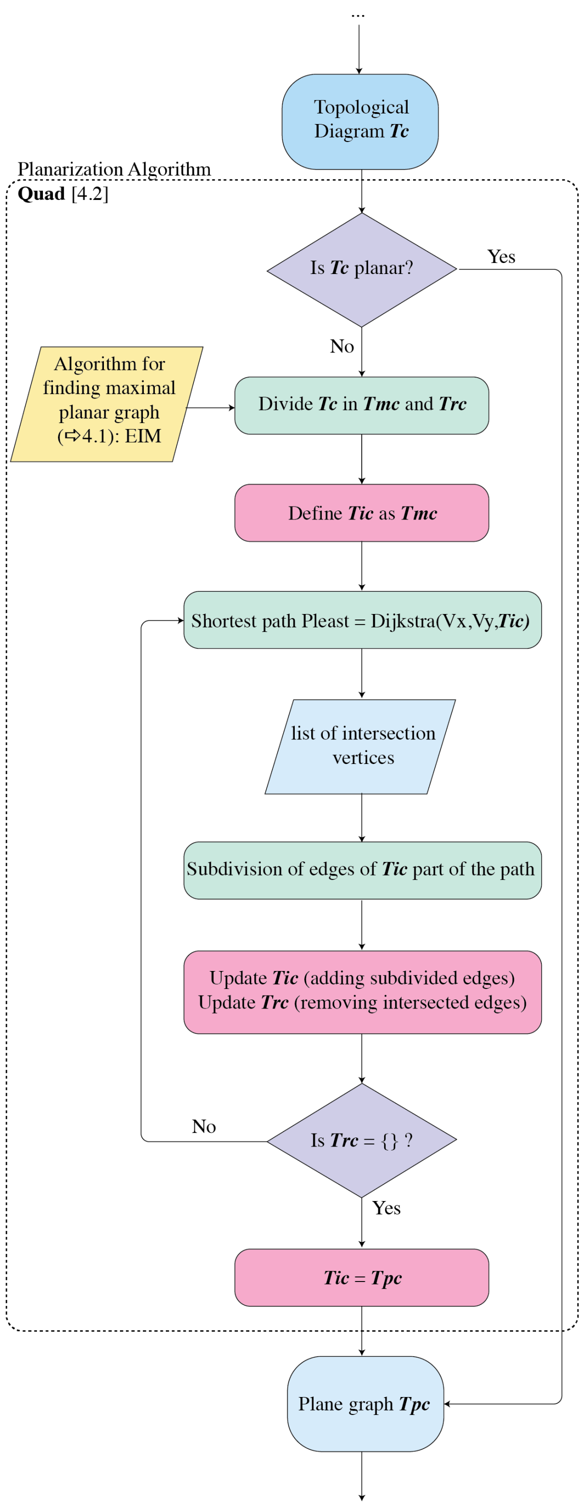


Fig. 10: QUAD algorithm, overview of the algorithm

This procedure planarizes the graph $\boldsymbol{T}$ creating *auxiliary cycles of force vector as quads* (*D'Acunto et al. 2019*) which can be seen as a 3D extension of *Bow, 1873*. It mainly consists in adding a new auxiliary vertex v_{Di} at every crossing of edges in $\boldsymbol{T_c}$, resulting in the creation of quadrilateral auxiliary cycles $<V_{Di}>^*$ in $\boldsymbol{F^*}$.

As it is introduced in Section 4.2.1, the initial graph $\boldsymbol{T_c}$ is cut in two subgraphs (using the EIM): a maximum planar subgraph $\boldsymbol{T_{mc}}$ and the resulting other subgraph $\boldsymbol{T_{rc}}$ (meaning $\boldsymbol{T_{mc}} + \boldsymbol{T_{rc}} = \boldsymbol{T_c}$). The iterative process modifies $\boldsymbol{T_{ic}}$ (corresponding to $\boldsymbol{T_{mc}}$ at the first iteration), adding back edges from $\boldsymbol{T_{irc}}$ (corresponding to $\boldsymbol{T_{rc}}$ at the first iteration) so that $\boldsymbol{T_{ic}}$ is still planar and eventually contains all the initial edges of $\boldsymbol{T}$. The main challenge is defining how the edges of $\boldsymbol{T_{irc}}$ should be intersected with the ones in $\boldsymbol{T_{ic}}$, aiming to generate the least intersections possible. This issue can be regarded as finding the shortest path (in terms of visited vertices) between the two extremities v_x and v_y of the edge $e_{x\text{-}y}$ that is added back from $\boldsymbol{T_{irc}}$ to the embedded graph $\boldsymbol{T_{ic}}$. It can be regarded as the path that passes by the smallest number of faces (a face being a closed cell of the graph). The Dijkstra algorithm is used to find this sequence P_{least} (*Dijkstra, 1959*).

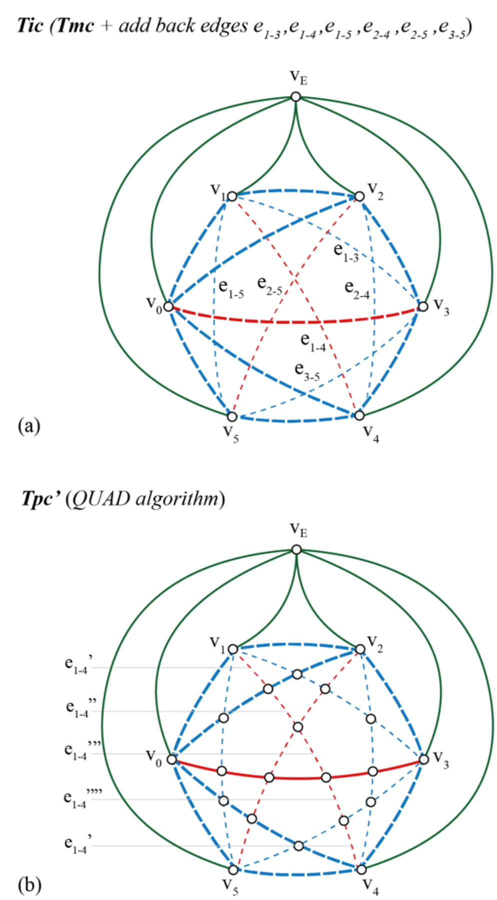


Fig. 11: Non-planar topological graph **Tic** composed of **Tmc+Trc** (thinner lines) resulting from EIM (left), and the resulting planar graph from the QUAD algorithm **Tpc** (right). The edges that were added back from **Trc** are represented in thin dotted lines.

4.2.1. Iterative intersection

Once the shortest path P_{least} = Dijkstra (v_x, v_y, $\boldsymbol{T_{ic}}$) is determined, the auxiliary intersection vertices can be found on the edges of ***Tic***. The intersections will split the edge $e_{a\text{-}b}$ separating two adjacent faces that are successive parts of the path. A new vertex labelled n_i ($_i$ being a numerical index) dividing $e_{a\text{-}b} \in$ ***Tic*** and $e_{x\text{-}y} \in$ ***Trc*** is introduced, resulting in two new pair of edges ($e_{a\text{-}b+}$, $e_{a\text{-}b++}$) and ($e_{x\text{-}y+}$, $e_{x\text{-}y++}$). The force cycle corresponding to the new vertex n_i is thus composed of the four forces vectors ($e_{a\text{-}b+}$, $e_{a\text{-}b++}$, $e_{x\text{-}y+}$, $e_{x\text{-}y++}$) and is, therefore, a quadrilateral cycle in ***F****. Each iteration ends when ***Tic*** is updated containing all the edges going from the two vertices that were added back from ***Trc***, while staying a planar graph. The vertex n_i is considered in the next iteration as part of ***Tic***, and the newly introduced edges can be intersected as any other edge of the graph. When all the edges of ***Trc*** have been added into ***Tic***, the updated graph includes all edges of ***T*** and is still planar. This resulting graph is ***Tpc***. The algorithmic implementation of this process takes benefit of algebraic calculation (face adjacency matrix) and is outlined in the Code Snippet 1.

Data:

T_{ic}: $(T_{ic}|E_{ic}, V_{ic}, F_{ic})$

T_{mc}: $(T_{mc}|E_{mc}, V_{mc}, F_{mc})$

T_{rc}: $(e_{i-j}^{rc} \in T_{rc})$ in which i and j are the index of the extremes of e_{i-j}^{rc}.

p_{i-j}^{rc}: A path comprises a sequence of f_{ic}. e_{i-j}^{rc} is supposed to cross over and cut the $e_{ic} \in E_{i-j}^{ic}$ into the segments E_{i-j}^{sp} following the p_{i-j}^{rc}.

M_{adj}^{f}: The face adjacency matrix of F_{ic}.

E_{i-j}^{rc}: The set of edges subdivided from e_{i-j}^{rc} according to the p_{i-j}^{rc}. Adding E_{i-j}^{rc} to E_{ic} is the same with adding e_{i-j}^{rc} back to T_{ic} and keep the graph T_{ic}' still planar.

E_{i-j}^{sp}: The set of edges from E_{ic} and subdivided by e_{i-j}^{rc} into segments $\{e_k^{sp}, \dots, e_q^{sp}\}$.

Input: T_{mc}, T_{rc}

Output: T_{pc}

$T_{ic} \leftarrow T_{mc}$

foreach e_{i-j}^{rc} in T_{rc}:

    $M_{adj}^{f} = T_{ic}.CreateFaceAdjMatrix()$;

    $f_i^{ic} \leftarrow T_{ic}.GetAdjacentFace(\mathrm{v}_i)$;

    $f_j^{ic} \leftarrow T_{ic}.GetAdjacentFace(\mathrm{v}_j)$;

    $P_{i-j}^{rc} \leftarrow \{\}$;

    $p_{i-j}^{rc} \leftarrow M_{adj}^{f}.\mathrm{ShortestPath}(f_i^{ic}, f_j^{ic})$;

    $E_{i-j}^{rc} \leftarrow p_{i-j}^{rc}.SplitEdge(e_{i-j}^{rc})$;

    $E_{i-j}^{sp} \leftarrow p_{i-j}^{rc}.SplitEdge(E_{i-j}^{ic}|e_{ic} \in E_{ic})$;

    $T_{ic}.E_{ic}.Remove(e_{ic})$;

    $T_{ic}.E_{ic}.AddRange(E_{i-j}^{sp}\{e_k^{sp}, \dots, e_q^{sp}\})$;

    $T_{ic}.E_{ic}.AddRange(E_{i-j}^{rc}\{e_{i-n}^{sp}, \dots, e_{m-j}^{sp}\})$;

end

$T_{ic}.E_{ic}.UpdateWith(E_{ic})$

$T_{ic}.F_{ic}.UpdateWith(E_{ic})$

$T_{pc} \leftarrow T_{ic}$

Code Snippet 01: QUAD algorithm

4.2.2. Force vectors corresponding to the labelled edges

As mentioned in the literature (*D'Acunto et. al, 2018*), in graphic statics to each edge of ***F*** correspond two opposite force vectors in ***F****, belonging to the two closed cycles of force vectors related to both extremity vertex of the selected edge of ***F***. In this case, an edge of ***F*** might appear in more than two forces cycles, since it can be split again at each iteration resulting for instance in ($e_{i\text{-}j++}$, $e_{i\text{-}j+++}$) $\in$ ($e_{i\text{-}j+}$,..., $e_{i\text{-}j++++}$)|$e_{i\text{-}j}$ $\in$ ***Tic***. The quadrilateral cycle <v_{Di*}> that are introduced with the present method are always constituted of two pairs of opposite vectors. In this example, the force vectors ($e_{i\text{-}j++}$, $e_{i\text{-}j+++}$) from ($e_{i\text{-}j+}$, $e_{i\text{-}j++++}$) and be calculated with the rule of $+v_{st}(e_{i-j+}) = -v_{ed}(e_{i-j+}) = +v_{st}(e_{i-j++}) = -v_{ed}(e_{i-j++}) = +v_{st}(e_{i-j+++})$ … the label of the edge provide necessary information to find the force vector of the segmented auxiliary edges of ***Tpc***.

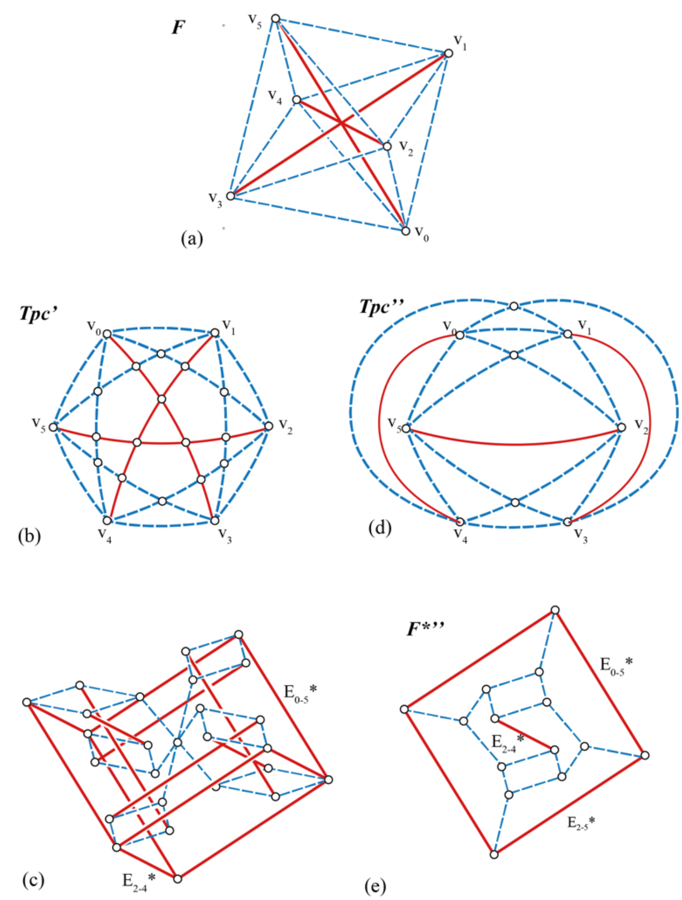


Fig. 12: Form diagram ***F***, Graph ***Tpc'*** (and corresponding Force Diagram ***F*'***) planarization procedure starting with ***Tmc*** generated by the EIM, Graph ***Tpc''*** (and corresponding Force Diagram ***F*''***) planarization procedure starting with ***Tmc*** generated by the VIM.

4.3. The MED procedure

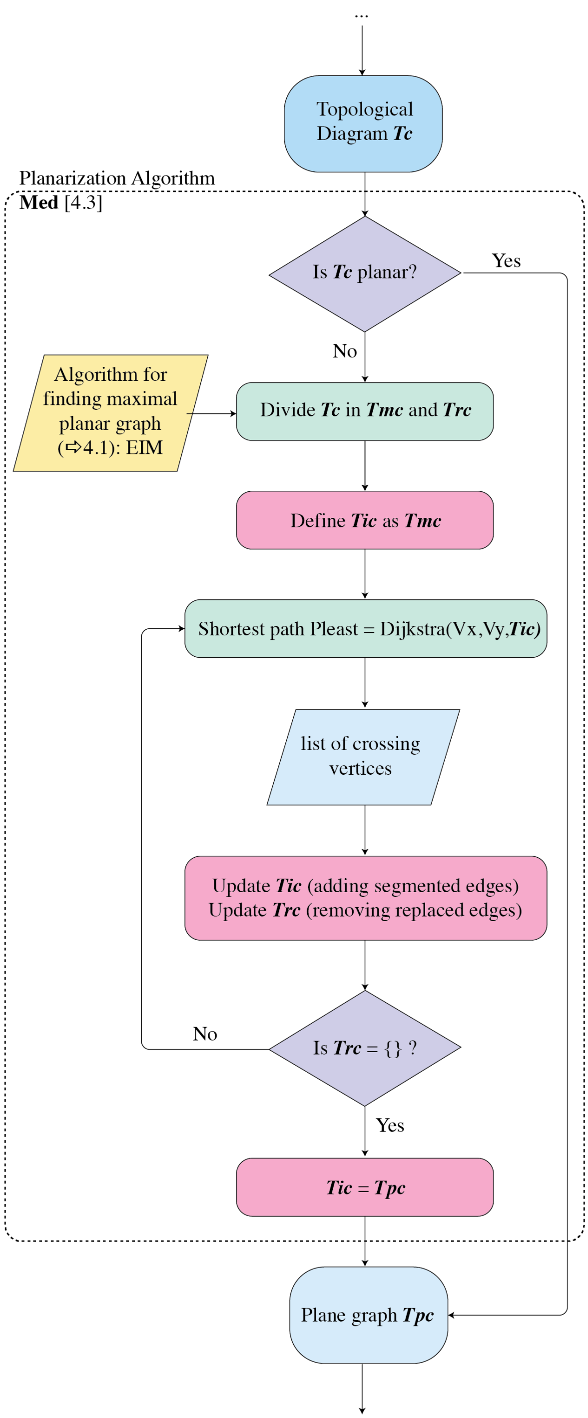


Fig. 13 : MED procedure, overview of the algorithm.

For this second algorithmic strategy, the developed method aims to add back edges from T_{rc} to T_{mc} without splitting the edges of T_{mc}. It takes the benefit of the existing vertices of the graph to subdivide the re-introduced edges from T_{rc}. Concretely, the edge re-introduced from T_{rc} to T_{mc} is decomposed as a series of edges visiting existing vertices of the graph that are located on the shortest path between its two extremities vertices.

4.3.1. Least edge splitting

The procedure to add edges from T_{rc} to T_{ic} consists in finding the shortest path through the existing vertices of T_{ic}. For an edge $e_{a\text{-}b}(v_a,v_b)$ from T_{rc} that is added back to T_{ic}, The shortest path is found thanks to a Dijkstra algorithm (*Dijkstra, 1959*) $P_{least}'(e_{a\text{-}b})$ = Dijkstra(v_a,v_b,T_{ic}) that returns a list of vertices $\{v_a,v_x,v_y...v_b\}$. T_{ic} is updated with all the edges connecting a vertex to the next one in this list, labelled as $\{e_{i\text{-}j}',e_{i\text{-}j}'',e_{i\text{-}j}'''...\}$. In this approach, a force vector is repeated more than twice in F^* (which is also the case for the QUAD algorithm described above). The number of occurrences corresponds to the length of the list given by Pleast'(v) minus one. In any case, this does not affect the equilibrium of the structure. The planarity is eventually achieved because of the fundamental principle of the MED algorithm. Since an edge is decomposed into a series of edges connecting each time neighbour vertex, the consequence is that the crossing of edges is avoided by definition.

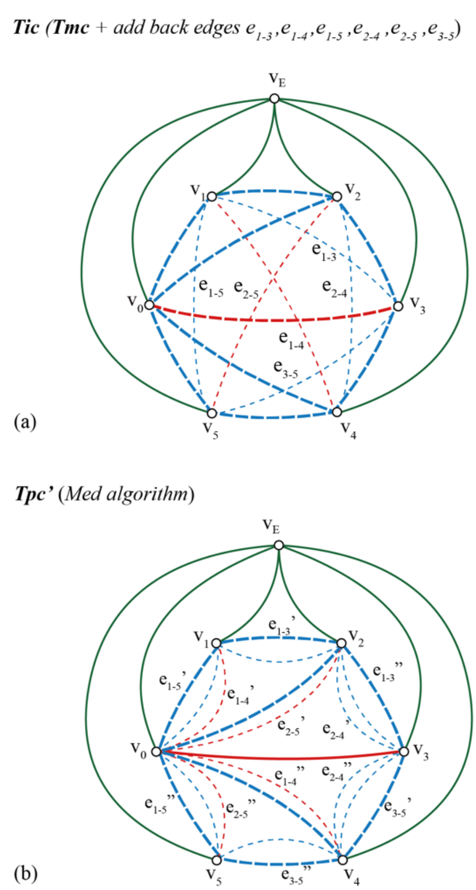


Fig. 14 : Non-planar topological graph ***Tic*** composed of ***Tmc***+***Trc*** resulting from EIM (a), and the resulting planar graph from the MED algorithm ***Tpc*** (b). The edges that were added back from ***Trc*** are represented in thin dotted lines.

Similarly to the QUAD algorithm, the algorithmic implementation of this process takes benefit of algebraic calculation (face adjacency matrix) and is outlined in the Code Snippet 2.

Data:

T_{ic}: $(T_{ic}|E_{ic}, V_{ic})$

T_{mc}: $(T_{mc}|E_{mc}, V_{mc})$

T_{rc}: $(T_{rc}|e^{rc}_{i-j})$ in which i and j are the index of the extremes $\{v_i, v_j\}$ of e^{rc}_{i-j}.

p^{rc}_{i-j}: A path comprises a sequence of v_{ic}. e^{rc}_{i-j} is supposed to be segmented into $\{e^{sp}_{i-n}, \dots, e^{sp}_{m-j}\}$ via $\{v_n, \dots, v_m\}$ following the p^{rc}_{i-j}.

M^{v}_{adj}: The adjacency matrix of V_{ic}.

E^{rc}_{i-j}: The set of edges subdivided from e^{rc}_{i-j} according to the p^{rc}_{i-j}. Adding E^{rc}_{i-j} to E_{ic} is the same with adding e^{rc}_{i-j} back to T_{ic} and keep the graph T'_{ic} still planar.

Input: T_{mc}, T_{rc}

Output: T_{pc}

```
T_ic ← T_mc
M^v_adj = T_ic.CreateVertexAdjMatrix();
foreach e^rc_{i-j} in T_rc:
    v^ic_i, v^ic_j ∈ V_ic;
    p^rc_{i-j} ← M^v_adj.ShortestPath(v^ic_i, v^ic_j);
    E^rc_{i-j}{e^sp_{i-n}, ..., e^sp_{m-j}} ← p^rc_{i-j}.Split(e^rc_{i-j})
    T_ic.E_ic.AddRange(E^rc_{i-j});
end
T_pc ← T_ic
```

Code Snippet 02: *MED* algorithm

4.3.2. Cycle organization

While adding the segments of the edges {e_{ij}',e_{ij}'',e_{ij}'''} to the vertices v_i,…,v_j, there is a principle to follow to keep the embedded graph ***Tic*** still planar. It is known that P_{least}'(e_{i-j}) has determined the vertices that the different segments of the edge{e_{ij}',e_{ij}'',e_{ij}'''} will connect. For instance, the vector e_{ij}'' that would connect v_x and v_y is to be added into both vertices cycles: e_{i-j}''(v_x,v_y)|{C_x, C_y}. As it is ruled in the embedded graph, each node has a clockwise sequence order, and thanks to this and the way edges are inserted back in the graph, the latter doesn't have an intersection among the edges (and thus stays planar). When the edge e_{i-j}'' is to be added in-between the adjacent vertices {v_x,v_y} of the embedded graph, it is important that in the two cycles C_x{…,e_{x-y},…}, C_y{…,e_{x-y},…} the edge $e_{i-j''}$ be inserted invertedly in both sequences so that C_x{…,$e_{i-j''}$,e_{x-y},…} and C_y{… ,e_{x-y}, $e_{i-j''}$…}.

*Conceptual difference between QUAD and MED, the impact of the algorithmic definition on the configuration of the resulting force diagram **F****

Fig. 12 and 15 apply respectively the QUAD and MED algorithm to the same case-study, i.e. the planarization of the topological graph ***T*** of an externally loaded octahedron. In both cases, the initial ***T*** = ***T_{mc}*** + ***T_{rc}*** is obtained after application of the EIM. The result of these two figures illustrates very clearly the conceptual difference of both algorithms.

Because the information to assemble the force diagram ***F**** is extracted from the resulting ***T_{pc}***, it is expectable that they produce different typologies of ***F****. A comparison of fig 12 and 15 (left column) shows the consequence in ***F****. The "quads" can be easily identified in the drawings (c) and (e) from Fig. 12 and are a direct consequence of the algorithmic definition. In a similar way, the closed cycle of forces in the drawings (c) and (e) from Fig. 15 can be identified and correspond directly to the multiple edges connected at each vertex in the graph planarized by the MED algorithm (respectively (b) and (d)).

Depending on the configuration of the initial form diagram ***F***, one of the two algorithms can provide a more readable force diagram. The definition of the planarization algorithm is thus crucial since it determines the configuration of ***F****, which visual aspect is central for graphic statics.

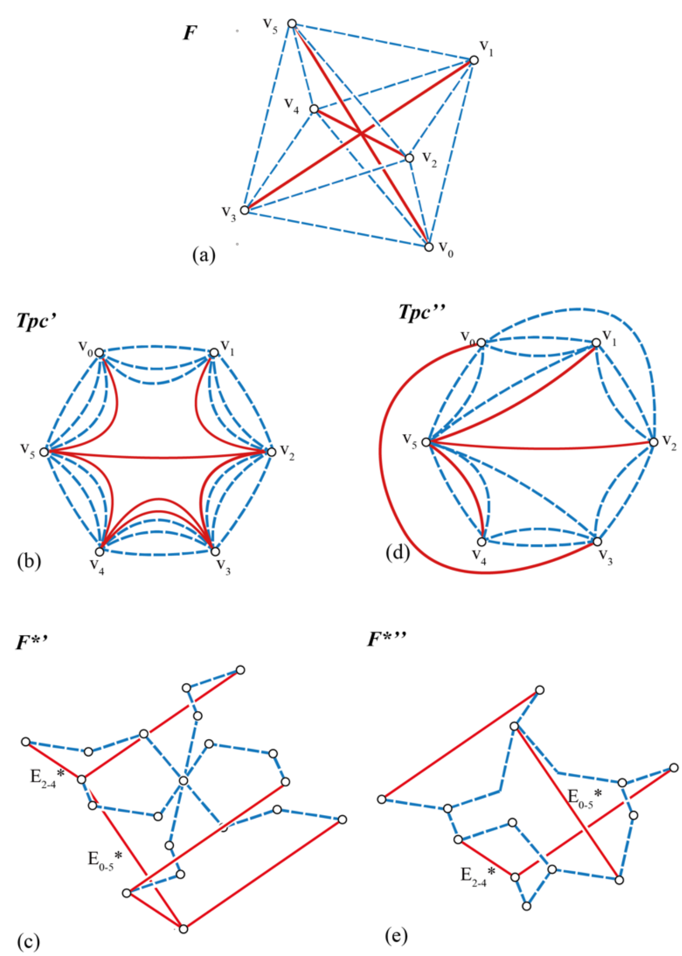


Fig. 15: (a) Form diagram ***F***, (b) Graph ***Tpc'*** (and (c) corresponding Force Diagram ***F*'***) planarization procedure (QUAD) starting with ***Tmc*** generated by the EIM, (d) Graph ***Tpc''*** (and (e) corresponding Force Diagram ***F*''***) planarization procedure (QUAD) starting with ***Tmc*** generated by the VIM.

5. The VGS-tool, implementation of the VGS algorithm as a CAD plug-in

5.1. Implementation

The algorithms presented in this paper are implemented into the VGS-tool, a plugin for the CAD environment Grasshopper3D in Rhino3D (McNeel, 2023) which is widely used by structural engineers and architects. The tool is written in C# and relies on two main libraries: the MathNet.Numerics (www.mathdotnet.com) for the algebraic operations and the C# implementation of the Boyer-Myrvold algorithm (*Boyer and Myrvold, 2004*) for the planarization operations. Some specific features of the VGS-tool are presented in sections 5.2 and 5.3

5.2. Real-time transformation of form and force diagrams

The interdependence between form and force diagrams in graphic statics allows transforming one of the diagrams while directly evaluating the consequent transformation of the other diagram (*D'acunto et al.,2017*) A set of geometric constraints must be defined to ensure the mutual dependence of ***F*** and ***F**** and thus guarantee the static equilibrium of the structure. Corresponding edges in the two diagrams (***F*** and ***F****) should be kept parallel to each other, while the vectors of the non-overlapping pairs of ***F**** (for diagrams ***F*** with underlying non-planar graphs) should as well maintain an equal length. The simultaneous constraint transformations of ***F*** and ***F**** can be achieved by means of numerical simulations, such as the Kangaroo2 (Piker, 2023) plug-in used within the McNeel Grasshopper3D and Rhino3D (McNeel, 2023) native environment. Besides the integrated constraints defined above, optional geometric constraints can be applied to ***F**** and ***F*** to fulfil specific design conditions. Thanks to the real-time transformations of form and force diagrams in the VGS-tool, the adjustment of the magnitude and direction of forces in vector-based 3D force diagrams can be used as an active operation in the structural design process as well as a geometrical constraint regarding the structure itself. An example of this transformations can be appreciated in Fig. 16

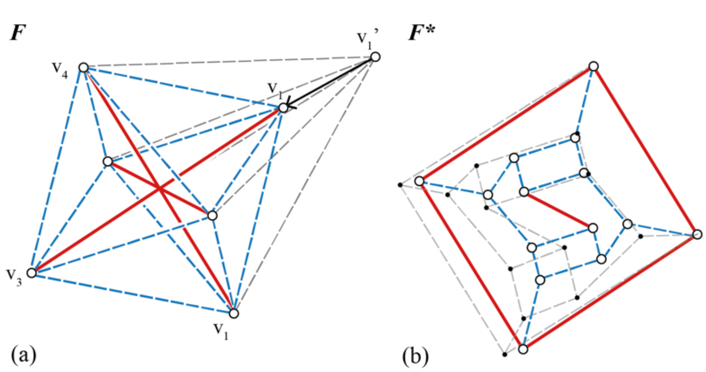


Fig. 16: Interdependent real-time transformation of form diagram *F* (a) in dashed lines, and force diagram *F** (b) in dashed line, with the result in the form diagram ***F*** (a) in colors, and in the force diagram ***F**** (b), in colors.

5.3. Form-finding of geometries at equilibrium

As explained in the previous sections, for a given structure ***F*** in static equilibrium, the VGS-tool can automatically produce a suitable planarized version of the underlying graph ***Tp*** (see Section 4) that is used to generate the closed cycle of force vectors that make up the corresponding vector-based force diagram ***F****. By imposing specific constraints to vector-based form and force diagrams – i.e. parallelism between corresponding edges in the two diagrams and equivalence in length and parallelism between duplicate edges in the force diagram – the two diagrams can undergo interdependent transformations while preserving the static equilibrium of the structure (Section 5.2).

Thanks to this transformation function, the VGS-tool can also be used to impose static equilibrium to an initial form diagram ***F*** that is not already at equilibrium. In this case, a default distribution of tension and compression forces is initially assigned to the edges of the form diagram ***F***. After defining a planarized version of ***Tp***, a set of open cycles of vectors based on the default distribution of forces is first generated. While imposing the transformations mentioned above, it simultaneously forces the vectors' cycles to close. The force vectors are consequently iteratively modified by the transformation algorithm by changing their lengths and lines of action until a valid vector-based force diagram ***F**** is generated. Since the vector-based diagrams are interdependent, the geometry of ***F*** is simultaneously modified to obtain a structure in static equilibrium. This feature of the VGS-tool allows to solve the static equilibrium, but it also makes it possible to find the form of structures in equilibrium independently of other structural form-finding tools. For instance, this feature can be very useful for the design of funicular and tensegrity structures among others. It can also be used to solve the static equilibrium of a given structure without any specific equilibrium calculation.

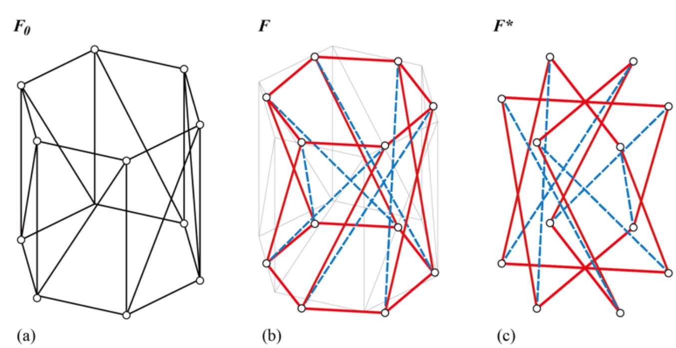


Fig. 17: Form-finding of a tensegrity structure. (a) the initial geometry F0, (b) the form-found Form diagram F and (c) the corresponding force diagram F*.

5.4. Organization of the VGS-tool

The plugin is organized into four toolsets, *Assemble structure, Generate diagrams, Transformation Visualization*. It covers the whole general process described in Section 3.1.

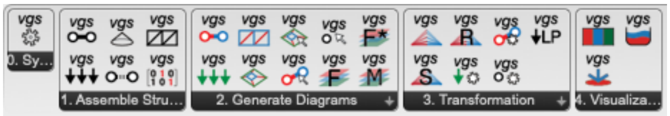


Fig. 18: Image of the toolbar of the VGS-tool in the workspace of Grasshopper3D – part of Rhino3D (McNeel, 2023).

The first part (*Assemble structure*) includes the modules that allow the assembly of the structural model consisting of the edges of the structures, the supports, the applied loads, and eventual self-stress. If necessary, the equilibrium calculation is performed with the *Evaluate Equilibrium* module with a numerical solver.

The second part (*Generate diagrams*) is the core of the tool. It assembles the form and the force diagrams. The main algorithm developed in Section 4 that planarizes the graph and assemble is contained in the "Assemble Force Diagram" module. In the first release of the tool, the user cannot choose the planarization algorithm. The script only implements the QUAD procedure (see Section 4.2.2). Another functionality allows the user to planarize the graph manually and integrate it from a geometrical drawing in the rhinoceros interface. The generated force diagram gives the user the option to see the fully assembled Force diagram (integrated ***F****) and a separate view of each node's force cycles (discrete ***F****).

The third part (*Modify diagrams*) allows the user to modify one of the diagrams and simultaneously assess the resulting modification on the other diagram. These parts make use of the numerical solver Kangaroo2 *(Piker, 2023)* included in Grasshopper3D – part of Rhino3D version 7 (*McNeel, 2023*).

The fourth part (*Visualization*) includes the modules for the graphical visualization of the diagrams. It allows the user to modify the visualization parameters (line thickness with respect to the magnitude of the forces and the label size) and export data.

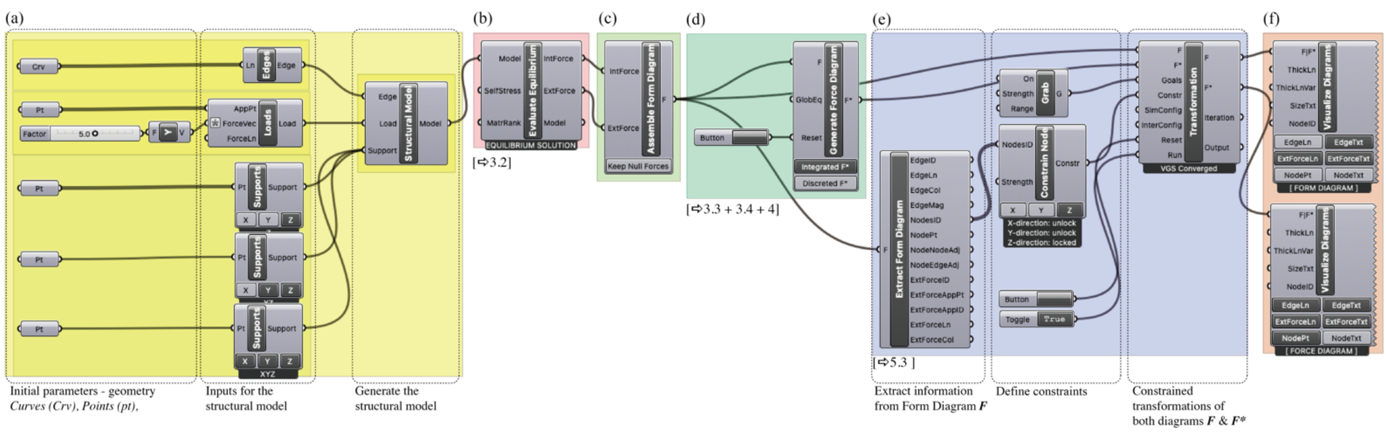


Fig. 19: Image of an entire definition of a structure in VGS-tool in the workspace of grasshopper. (a) definition of structural model (yellow), (b) evaluation of equilibrium (red), (c) assembly of form diagram (light green), (d) generation of force diagram (dark green), (e) transformations (blue), (f) visualization of the results (orange)

5.5. Case study

This section presents a conceptual design case study to demonstrate the potential application of the proposed development to realistic design scenarios. The structural concept for a stadium roof is laid out according to a fictitious design scenario based on the Stadium in Braga by architect Eduardo Souto de Moura and engineer Rui Patricio. Specifically, the symmetric roof is placed only on the pitch's two longitudinal sides and is conceived as a spatial tied-arch system. Both tied-arch systems work as a unit since their ties are connected with horizontal cables that span over the short side of the pitch. Each arch is connected to its corresponding tie with x-braced cables and is furthermore stabilized with additional x-braced cables that form a façade-like cable-net on the back side of the stands. The support points of the tied arch are 150 m apart from each other. Only the self-weight of the arch and a constant force of (3'000 kN) in the horizontal cables are considered during this early-stage form finding.

A preliminary 3D form diagram $\boldsymbol{F_a}$ and the geometric constraints can be seen in Fig. 20 (a) This first instance has been generated using the Combinatorial Equilibrium Modelling (CEM) form-finding method (*Ohlbrock and D'Acunto, 2020; Ohlbrock et al., 2017*). Apart from the equilibrium condition, which is a hard constraint in the CEM formulation, additional constrained planes have been activated (*Pastrana et al. 2023*) to keep the segmentation of the structure as desired. It can be easily seen that the initial equilibrium state does not fulfil the geometric constraints of the support points.

The VGS-tool has generated the corresponding force diagram ***F****, based on which the transformation module (using Kangaroo2) has been used to transform the equilibrium state. Thus, the tool has been used to match the given support points while keeping the corresponding edges in the two graphs (E_{i-j} and E_{i-j}*) parallel to each other and ensuring that the vectors of the non-overlapping pairs of ***F**** are kept parallel and equal in length. The resulting form $\boldsymbol{F_b}$ and force diagram $\boldsymbol{F_b}$* can be seen in Fig.

20.

The setup has been used to evaluate two alternative solutions in a further step. The first alternative was triggered through the idea to adapt the forces in the tie to be closer in magnitude to the ones in the arch. Consequently, the force diagram $\boldsymbol{F_c^*}$ undergoes a local transformation (in the y-direction). At the same time, the shape of the arch $\boldsymbol{F_c}$ and the tie pre-dominantly reacts with a change in the other two dimensions (in the x- and z-directions).

A second alternative was generated through a targeted change in the horizontal forces in the cables. More specifically, the forces have been increased by 40%. Consequently, the force diagram $\boldsymbol{F_d^*}$ undergoes a local transformation (in the x-direction), resulting in a tie geometry $\boldsymbol{F_d}$ with a larger sag (in the x-direction).

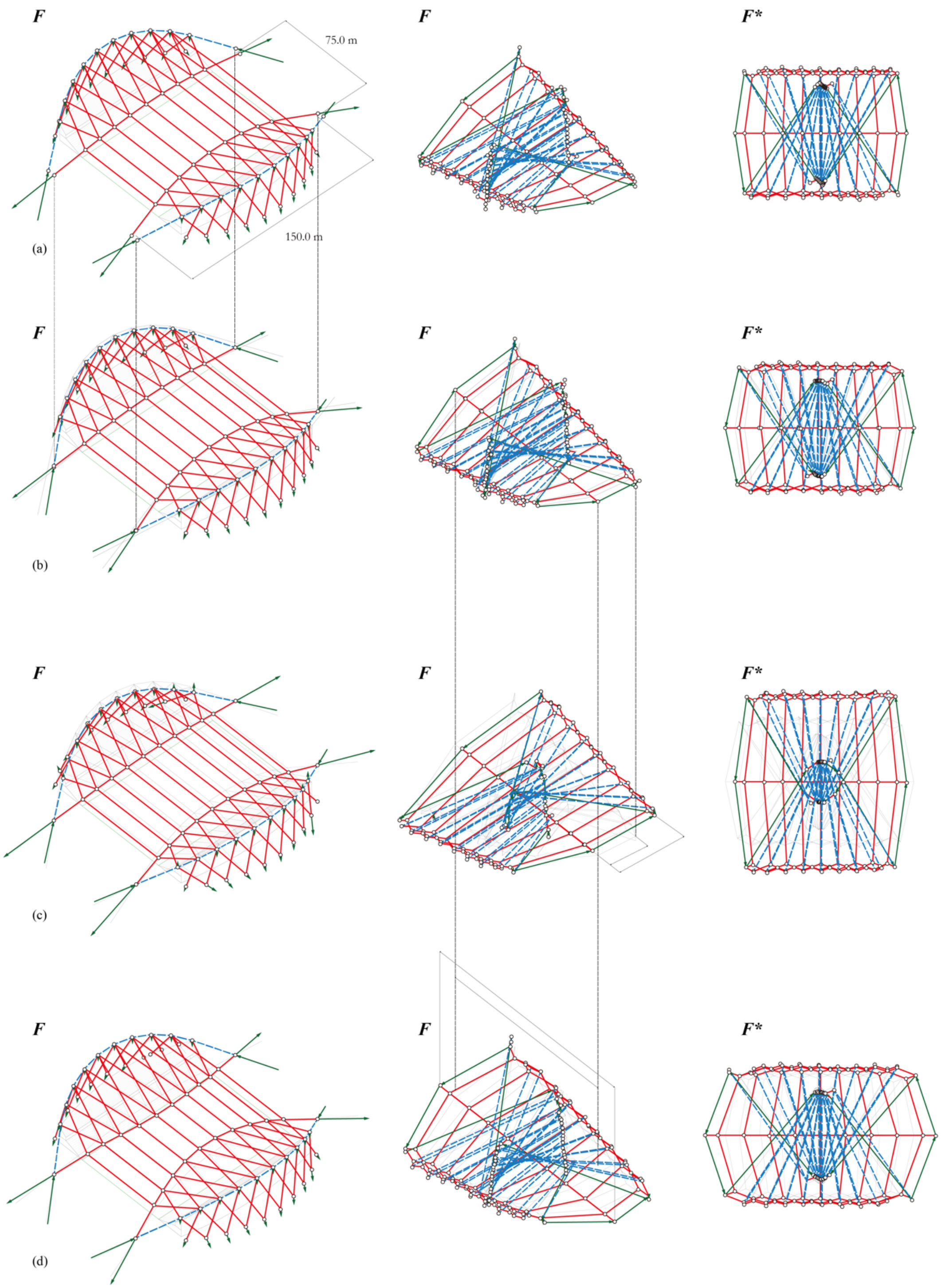


Fig. 20: Case study: structural design of a stadium roof. (a) initial Form diagram ***F*** with corresponding force diagram ***F**** seen from two different points of view. (b) the first transformation leads to the merging of the support points (c) the second transformation equalizes the forces in the main cable and in the arch..(d) the third and last transformation increases the forces in the horizontal cables over the pitch.

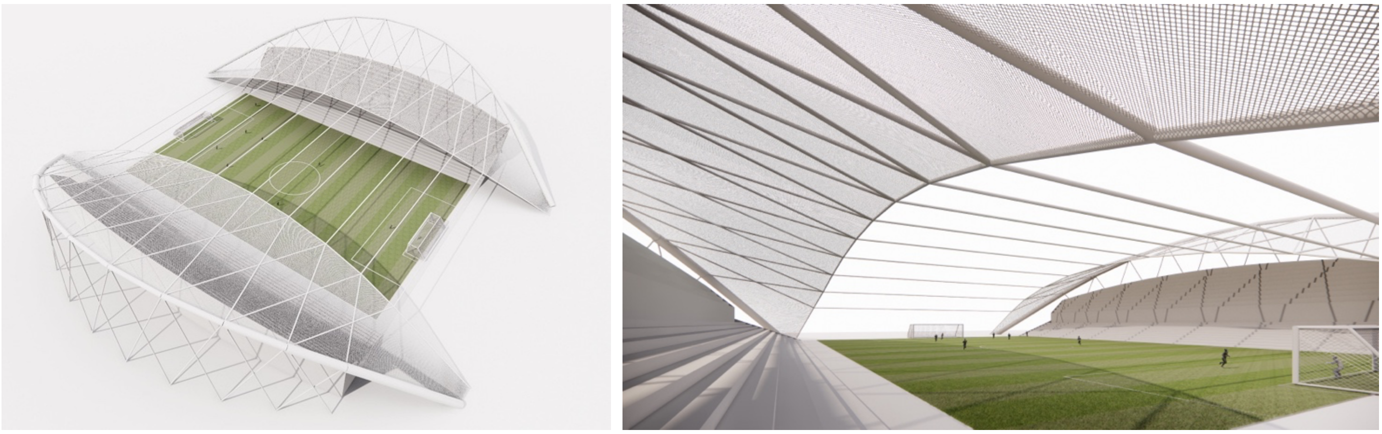


Fig. 21: Case study: structural design of a stadium roof. Left: view from top; right, view from under the structure.

6. Conclusions

This paper introduced a novel graphic statics computational implementation for the automated generation of vector-based form and force diagrams for both 2D and 3D structures. The "global" algorithm can solve any structure typology in static equilibrium, mixing both tension and compression elements that are either externally loaded or self-stressed.

The contribution develops in detail two novel algorithms that planarize the topological diagram so that a force diagram for a structure with a non-planar underlying graph can be assembled. The associated data structure implemented in the computational procedure enables dealing effectively with structures composed of a large number of elements.

Furthermore, the paper explained a method allowing real-time transformations of form or force diagrams using the numerical simulation Kangaroo library. This feature that was first introduced in *D'acunto et al., 2017* can either facilitate the modification of an initial force diagram to fulfil specific geometrical requirements allowing the designer to evaluate in real time the consequences of such modifications in terms of force magnitudes, orientation, and distribution. It can also be used to modify the forces to attain specific goals in terms of directions, intensity, distribution, and structural behaviour (tension/compression), while visualizing the affected geometry in real-time. Some sets of specific constraints can also be applied to this transformation regarding the form and force diagram (geometrical domains, maximal or minimal values of lengths/force intensities/…). Additionally it can be used to solve the static equilibrium of a given structure, which also proves useful to form-find geometries at equilibrium.

7. Limitations and future work

Further investigation will focus on three main topics.

First, the visualization of 3D diagrams on a flat screen always represents a difficult task. Future developments will focus on integrating specific visual effects that help the user to read more directly the depth of the diagram and consequently perceive better the length and angles of the vectors. Integrating specific visualisation using augmented reality could be of great use to address this issue as well in the next steps.

Secondly, tailored force diagram configurations for specific structural typologies will be defined and implemented into the VGS-tool. At this stage of the research, the paper presented two algorithms (the QUAD and the MED) that both generate different arrangements of force diagrams. An area of future research is to focus on specific assemblies of force diagrams based on a hierarchical organization. This could help the user to activate specific sub-parts. That is first related to more theoretical/fundamental research before its algorithmic implementation (i.e. related to graph theory and planarizing methods). Moreover, an extra algorithm that scans the structures and identifies typologies and/or hierarchy of its arrangement could be introduced before the planarization of the graph. The results would then inform the planarization process to choose between one of the available planarization algorithms in which the process is specifically developed for such typology.

Eventually the overall algorithmic procedure will be monitored and benchmarked to look for optimization. Nevertheless, the presented framework gives efficient results and computing times that are more than acceptable for the user. A more thorough set of tests recording performance with slightly different variants in implementation of the planarization algorithm will help identify how to improve the performance of the algorithm.

8. References

1. Akbarzadeh, M. "3D Graphic Statics Using Polyhedral Reciprocal Diagrams." Diss. ETH Zurich, 2016.
2. Lee, J. "Computational Design Framework for 3D Graphic Statics." Diss. ETH Zurich, 2019.
3. Beineke, L. W., and R. E. Pippert. "Planar Graphs." Bulletin of the American Mathematical Society, vol. 84, no. 5, 1978, pp. 766-

782.

4. Beghini, L., et al. "Structural Optimization Using Graphic Statics." Structural and Multidisciplinary Optimization, vol. 49, no. 3, 2014, pp. 351-366.
5. Buchheim, C., et al. "Crossings and Planarization." Handbook of Graph Drawing and Visualization, edited by R. Tamassia, CRC Press, 2013.
6. Boyer, J., and W. Myrvold. "On the Cutting Edge: Simplified O(n) Planarity by Edge Addition." Journal of Graph Algorithms and Applications, vol. 8, no. 3, 2004, pp. 241-273.
7. Bow, R. "Economics of Construction in Relation to Framed Structures." E. and F.N. Spon, London, 1873.
8. Brandes, U. "Efficient Planar Embedding of Sparse Graphs." ACM Transactions on Algorithms, vol. 6, no. 4, 2000, pp. 568-579.
9. D'Acunto, P., et al. "Vector-based 3D Graphic Statics: A Framework for the Design of Spatial Structures Based on the Relation Between Form and Forces." International Journal of Solids and Structures, vol. 167, 2019, pp. 58–70.
10. D'Acunto, P., et al. "Vector-based 3D Graphic Statics: Transformation of Force Diagrams." Proceedings of the IASS 2016, Tokyo.
11. Dijkstra, E. W. "A Note on Two Problems in Connexion with Graphs." Numerische Mathematik, vol. 1, no. 1, 1959, pp. 269-271.
12. Fivet, C., and D. Zastavni. "The Salginatobel Bridge Design Process by Robert Maillart (1929)." Journal of the International Association for Shell and Spatial Structures, vol. 53, 2012.
13. Hagberg, Aric, Swart, Pieter J., and Schult, Daniel A.. "Exploring network structure, dynamics, and function using NetworkX." Proceedings of the 7th Python in Science Conference, 2008, Pasadena.
14. Harary, F. "Graph Theory." Addison-Wesley, 1969.
15. Jasienski, J.P., et al. "Vector-based 3D Graphic Statics (Part II): Construction of Force Diagrams." Proceedings of the IASS 2016, Tokyo.
16. Lee, J., T. Van Mele, and P. Block. "Disjointed Force Polyhedra." Computer-Aided Design, vol. 99, 2018, pp. 11–28.
17. Mackinlay, J. "Automating the Design of Graphical Presentations of Relational Information." Transactions on Graphics, vol. 5, no. 2, 1986, pp. 110–141.
18. Maxwell, J.C. "On Reciprocal Figures, Frames, and Diagrams of Forces." Philosophical Magazine, vol. 27, 1864, pp. 250–261.
19. Maxwell, J.C. "On Reciprocal Figures, Frames, and Diagrams of Forces." Transactions of the Royal Society of Edinburgh, vol. 26, 1870, pp. 1–40.
20. McNeel R. and others. "Rhinoceros 3D", Version 7. Robert McNeel & Associates, Seattle, WA. 2023.
21. Milošević J. and Graovac O., "An Approach to Designing Architectural Structures Using 3D Graphic Statics » Advances in Architectural Geometry 2023, Berlin, Boston: De Gruyter, 2023, pp. 427-440.
22. Ohlbrock, P.O., and P. D'Acunto. "A Computer-Aided Approach to equilibrium Design Based on Graphic Statics and Combinatorial Variations." CAD Journal, vol. 121, 2020, 102802.
23. Ohlbrock, P.O., et al. "Constraint-driven Design with Combinatorial Equilibrium Modelling." Proceedings of IASS Annual Symposium, 2017, Amsterdam.
24. Pellegrino, S., and C.R. Calladine. "Matrix Analysis of Statically and Kinematically Indeterminate Frameworks." International Journal of Solids and Structures, vol. 22, 1986, pp. 409–428.
25. Pastrana, R., et al. "Constrained Form-Finding of Tension–Compression Structures Using Automatic Differentiation." Computer-Aided Design, vol. 155, 2023, 103435.
26. Piker, D.. "Kangaroo2", part of McNeel Grasshopper 3D in Rhino3D, Robert McNeel & Associates, Seattle, WA. 2023.
27. Pirard, A. "La Statique Graphique." Imprimerie Vaillant-Carmanne, 1950.
28. Pottman, H., et al. "Architectural Geometry." Bentley Institute Press, 2007.
29. Rankine, W.J.M. "A Manual of Applied Mechanics." Richard Griffin and Company, 1858.
30. Rankine, W.J.M. "Principle of the Equilibrium of Polyhedral Frames." The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, vol. XXVII, 1864, p. 92.
31. Sauer, R. "Differenzengeometrie." Springer, 1970.
32. Saviotti, C. "La Statica Grafica, Seconda Parte: Forze Esterne." Ulrico Hoepli, 1888.
33. Tamassia, R. "Handbook of Graph Drawing and Visualization." CRC Press, 2013.
34. Tarjan, R.E. "An Algorithm for Planarity Testing of Graphs." Journal of Computer and System Sciences, vol. 3, no. 2, 1970, pp. 110–118.
35. Van Mele, T., and P. Block. "Algebraic Graph Statics." Computer-Aided Design, vol. 53, 2014, pp. 104–116.
36. Van Rossum, G., and F.L. Drake. "Python 3 Reference Manual." CreateSpace, 2009.
37. Zalewski, W., and E. Allen. "Shaping Structures: Statics." Wiley, 1998.
38. Zastavni, D. "The Structural Design of Maillart's Chiasso Shed (1924): A Graphic Procedure." Structural Engineering International, vol. 18, no. 3, 2008, pp. 247-252.

39. "Polyframe."*Food4Rhino*,https://www.food4rhino.com/en/app/polyframe; 2023 [accessed 20 May 2023]
40. "COMPAS 3GS." *GitHub*, https://github.com/BlockResearchGroup/compas_3gs; 2023 [accessed 20 May 2023]
41. "Rhinovault 2." *Food4Rhino*, https://www.food4rhino.com/en/app/rhinovault-2; 2023 [accessed 20 May 2023]
42. "3D Graphic Statics." *Food4Rhino*, https://www.food4rhino.com/en/app/3d-graphic-statics.
43. "Math.NET." http://www.mathdotnet.com; 2023 [accessed 20 May 2023]

Fundings

(The author) Yuchi Shen received support by the Natural Science Foundation of China (NSFC#52208010) and the China Postdoctoral Science Foundation (#2022M720716).