COMPUTER-AIDED DESIGN & APPLICATIONS, 2016
VOL. 13, NO. 3, 271–280
http://dx.doi.org/10.1080/16864360.2015.1114384

Generator of 2D Geometric Constraint Graphs

Adel Moussaoui and Samy Ait-Aoudia

Ecole Nationale Supérieure en Informatique, Algeria

ABSTRACT
In 2-dimensional geometric constraint solving, graph-based techniques are a dominant approach, particularly in CAD context. These methods transform the geometric problem into a graph which is decomposed into small sub-graphs. Each one is solved, separately, and the final solution is obtained by recomposing the solved sub-graphs. To the best of our knowledge, there is no random geometric constraint graph generator so far. In this paper, we introduce a simple, but efficient generator that produces any possible geometric configuration. It would be parameterized to generate graphs with some desirable proprieties, like highly or weakly decomposable graphs, or restricting the generated graph to a specific class of geometric configuration. Generated graphs can be used as a benchmark to make consistent tests, or to observe algorithm behaviour on the geometric constraint graphs with different sizes and structural properties. We prove that our generator is complete and suitable for two main classes of solving approaches.

KEYWORDS
geometric constraint decomposition; graph-based solver; graph generator

1. Introduction

A geometric constraint system (GCS) consists of a finite set of geometric elements, such as points, lines and circles, along with relationships of different types such as distance, angle, incidence and parallelism between pairs of geometric elements. This problem is central in many applications, such as computer-aided design (CAD) [8], molecular modelling and recently localization in wireless sensor networks [22]. Solving a GCS consists of finding real coordinates of geometric elements in Euclidean space. If a GCS is incomplete, i.e. there are not enough constraints between geometric elements, then it is called under-constrained. If the specified constraints are conflicting, i.e. there are too many constraints defined between geometric elements, this situation is called over-constrained. A GCS is called well-constrained, if it has a finite number of solutions. We formally define those notions in the next section.

In this paper, we focus on geometric constraint systems in the generic sense; we focus only on the solvability of the graph and ignore the numerical values of the geometric constraint. Assigning coordinates to the geometric elements of a GCS that satisfies the constraints is called a realization problem [6], Saxe [20] has shown it to be NP-hard.

Many solvers have been proposed in the literature that can be classified in four broad categories: graph-based, symbolic, numerical and rule-oriented. Furthers details can be found in [3] and [12]. In this work, we focus on graph-based methods, developed in Computer-Aided Design context, such as those presented in [1], [2], [4], [10], [17], [19]. Many such solvers transform the GCS into a graph. By applying some decomposition techniques on the constraint graph, they isolate under, over, and well-constrained parts. The well-constrained part is then analyzed by a decomposition technique to find small solvable sub-graphs. The final solution is produced by merging the solved sub-graphs in respect to an order of resolution produced in the decomposition phase. This is referred to as Decomposition / Recombination plan (DR-Plan) [9]. The primary aim of this decomposition is to speed up the resolution process by limiting the use of direct algebraic resolution to subsystems that are as small as possible (typically ruler and compass solvable problems).

Hoffmann et al., [9], classify DR-planners into two main categories: SR-Planners (constraint shape recognition), and MM-planners (generalized maximum matching). Many SR-Planners have been proposed in the literature. In [15], Joan-Arinyo et al. introduce a solving approach, called tree decomposition, that recursively splits the constraint graph into three sub-graphs such that pairwise share one vertex (called hinge). Their method is based on searching for hinges in fundamental circuits

CONTACT Adel Moussaoui a_moussaoui@esi.dz

of a specific planar embedding of the constraint graph. Fudos and Hoffmann proposed in [4] an algorithm called reduction analysis which is based on a key concept called cluster. Initially, each edge of the constraints graph is considered as a clusters, the algorithm recursively merges each three clusters that share pairwise one vertex until one final cluster is obtained. Owen proposes in [9] an algorithm based on a recursive decomposition of a graph into tri-connected components. SR-Planners are not complete, i.e., they do not solve every well-constrained graph but only a sub-class of geometric configurations. These methods require that the constraint graph be bi-connected [15], and the smallest solvable sub-graphs are generally triangles [9]. Those algorithms can be theoretically extended to handle other types of shapes then triangles by giving a specific recognition algorithm to each new type of shape. We note that there is an infinite collection of shapes [13]. As a result, SR-planners cannot be generalized. The second class is referred to as MM-planner. In [2], Ait Aoudia et al. transform the GCS into a bipartite graph and use the maximum matching for finding solvable sub-graphs in the geometric constraint graphs. Ait Aoudia et al. proposed in [1] an algorithm that uses what they called a recursive skeletonization for decomposing the constraints graph. Hoffmann et al. proposed in [10] an algorithm called Frontier, which is based on the finding of minimal dense sub-graphs that are sequentially extended by adding more geometric objects one at a time. In MM-planner category, there is no restriction on the domain of geometric constraint configurations, the smallest sub-graph can be any non-reducible well-constrained graph. These categories detect smaller possible solvable sub-graphs and their order of resolution. Those methods are complete, i.e. any well-constrained graph can be decomposed, solved and recomposed (see Figure 1).

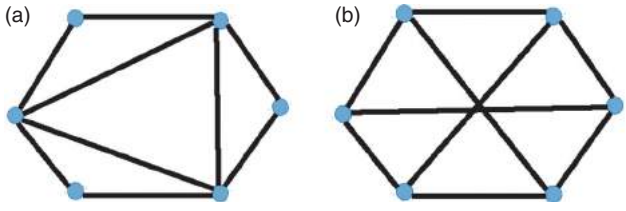

Figure 1. Two well-constrained graphs: (a) solvable by SR-Planners and (b) solvable only by MM-Planners.

In this paper, we discuss how to generate random geometric constraint configurations, with desired proprieties, that are in relation to the quality of the solver, described next in this paper. We focus on generating graphs for well-constrained geometric problems for which the Laman condition [16] holds.

Most of the decomposition algorithms proposed in the literature have been tested on small and well-chosen examples. However, to make consistent tests, or observe the algorithm behaviour on graphs with various sizes and structural properties, we propose the use of the graph generator that produces a set of geometric configurations, covering different situations of difficulty and sizes regarding the classification shown above. We have developed a graph generator that verifies some desired proprieties:

- **Completeness:** it can generate any possible geometric configuration.
- **Customizable:** it would generate some specific configuration. This is done by parameterizing the generator. It can be parameterized to produce a graph solvable by SR-Planners or only by MM-Planners. Moreover, it can generate a specific domain of GCS or a general one. By specifying what we called a degree of decomposition, we can build a graph that has a low or a high number of solvable sub-graphs. We can also specify the average size of the smallest sub-graph. Those two parameters represent our adopted metric of solvability.
- **Simplicity and efficiency:** our generator is easily understood; it is straight forward implemented based on a verified theory.

This paper is organized as follows: The next section highlight the application of geometric constraints solving in CAD software. Section 3 presents some notions relative to the geometric constraint graph, its decomposition and its generic solvability. Section 4 outlines the generation of random well-constrained graphs and presents some tests on their decomposability. Section 5 presents a suitable graph generator for well-constrained problems. Section 6 presents some experiments to evaluate the solvability of generated graphs. Finally, we summarize our contribution in section 7.

2. Application of Geometric Constraints Solving in CAD Software

Generally, in most CAD software, a 2D geometry is considered the starting-point for most 3D models. First, the designer sketches a 2D draft, then adds some relation between geometric elements. The geometric constraint solver transforms the problem into a graph of constraints. After the solving process, that can be done in real-time, geometric objects are adjusted to conform to the specified constraint. This offers the advantage of freeing the user from the tedious task of exact location of different geometric elements. The 2D sketch is extruded to

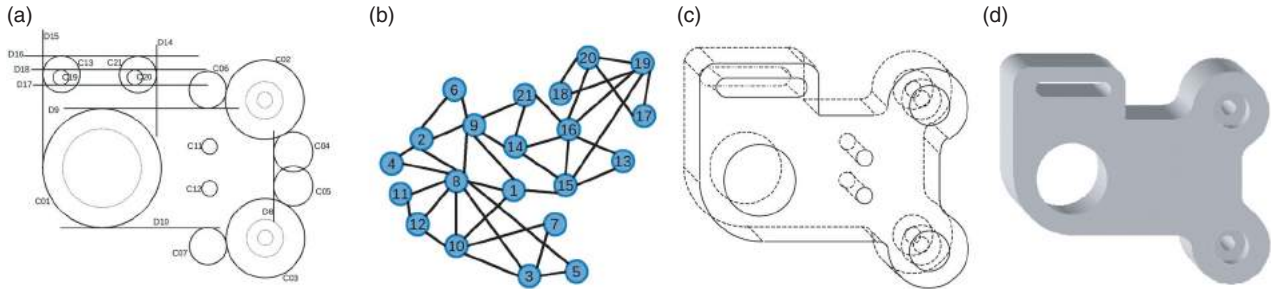

Figure 2. (a) a 2D constrained model, (b) the corresponding constraints graph, (c) the extruded model and (d) the final 3D model.

obtain the final 3D model. Geometric models will be easily updated in the future, by simply modifying the values of the different constraints. Because the underlying system of equation is non-linear, and most solving methods are $O(n^3)$ or worse, the resolution speed depends on the size of the largest system of equation. Decomposing the GCS, which is the central role of the planner, will speed up the resolution process and make the CAD software more interactive, thus the productivity of the designer will be increased.

To illustrate the importance of planners, we use the example of Fig. 2. The problem consists of 13 circles, 8 lines, which are connected by 19 tangency constraints: circle-circle and circle-line, 4 constraints of angles between lines and 16 distance constraints. The corresponding constraint graph is given in Figure 2(b), nodes are numbered from 1 to 21, and each constraint is represented by an edge of the graph. We have decomposed its corresponding graph using the method presented in [2]. The result gives 20 systems of equations, the largest of which contains 8 unknowns. Without the decomposition process, we have to solve a quadratic system of 42 unknowns. The time required for finding a solution will significantly decrease by decomposing the system into smaller subsystems, and consequently the design process becomes more interactive.

3. Geometric Constraint Graphs and Their Decomposition

Any GCS can be represented by a graph G, which consists of a vertex set V and an edge set E. The vertices of G represent the geometric elements, and the edges represent the constraints between them. The cardinality of V will be called the size of G.

3.1. Generic solvability of the geometric constraint problem

Laman theorem [16] gives the necessary condition of generic solvability for any GCS. To be solvable, its constraint graph must be structurally well-constrained, (also called generically isostatic, minimally rigid or Laman graph by rigidity theory and structural topology communities).

Definition 1: A geometric constraint graph $G = (V, E)$ where $|V| = n$ $(n > 1)$ and $|E| = m$ is structurally well-constrained if and only if $m = 2n - 3$ and $m' \leq 2n' - 3$ for any induced sub-graph $G' = (V', E')$, where $|V'| = n'$ $(n' > 1)$ and $|E'| = m'$.

Definition 2: A constraint graph $G = (V, E)$ contains a structurally over-constrained part if there is an induced sub-graph $G' = (V', E')$ having more than $2n'-3$ edges.

Definition 3: A geometric constraint graph $G = (V, E)$ is structurally under-constrained if it is not over-constrained and the number of edges is less than $2n - 3$.

Laman's theorem [16] is proved only for point to point distance constraints. There is no combinatorial characterization for any other geometric element [19]. The extension of his theorem to other geometric elements may imply incorrect cases. A typical example of such cases is given in [1]. The generalization of Laman's theorem to dimensions three, or higher, has been proved to be incorrect. Until now, there is no combinatorial characterization for 3D geometric constraint problems; it's an open problem.

3.2. Geometric constraint graph decomposition

Decomposition process can be done in two steps, the first one can be considered as a pre-processing that consists of isolating the unsolvable parts: the over- and under-constrained sub-graphs. For this purpose, we can use for example the algorithm proposed in [2]. Then, the obtained well-constrained graphs, which are the solvable ones, are decomposed in the second step. The goal of this decomposition is to speed-up the resolution process,

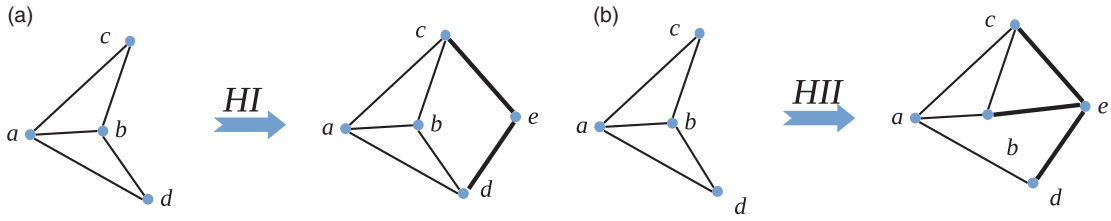

Figure 3. The two operation of Henneberg construction.

by providing a plan of resolution. We focus only on the decomposition of the well-constrained parts.

Transforming the whole geometric constraint problem into a system of equations, and solving it, is extremely time-consuming. The main goal of graph-based techniques is to decompose the problem into small subsystems. Each one is solved separately with respect to an order given as output by the DR-planer. Two parameters that we call a decomposability metrics can decide how fast will be the resolution process:

(1) the size of the largest sub-graph as proposed in [9] by Hoffmann et al.
(2) the number of detected sub-graphs over the total number of vertices. We can formulate this by a parameter, d, called the degree of decomposability, where $d = n/g$. n is the number of vertices of the geometric constraint graph and g is the number of detected sub-graphs to be solved directly by algebraic methods. d indicates if the graph is highly or weakly decomposable.

In the rest of this paper, we show how to design a random graph generator that uses those two parameters as input. Our approach is organized in two steps: in the first one, we present a procedure called *RH* that generates a non-decomposable graph, i.e., a graph that has a known size and no detectable sub-graphs. Then we use the procedure *RH* to develop our random graph generator, called *RRH*, which can produce a graph with a known size of the largest sub-graph and a known number of sub-graphs. *RRH* can also produce graphs that are solvable by SR-Planner or by only MM-Planners.

4. Generating Well-constrained Graphs

In [21], Tay and Whiteley presented an inductive construction that always produces a well-contained graph. This construction is due to Henneberg [7]. The definition follows.

Definition 4: Starting with a graph $G = K3$, a well-constrained graph can be built inductively by adding one vertex at a time using one of these two Henneberg operations:

> **Operation HI :** add a new vertex v to G, then connect v to two chosen vertices u and w from G via two new edges (v, u) and (v, w).
>
> **Operation HII :** add a new vertex v to G, chose an edge (u, w) and a vertex z from G, then add three edges (v, u), (v, w) and (v, z) to G, finally delete the edge (u, w).

Fig. 3 shows the two operation of Henneberg: (a) the first operation HI which consists of adding the vertex e and connecting it to two existing vertices by two edges: (e, c) and (e, d); (b) shows the second operation HII, it "split" the edge (b, d) by deleting it and adding a new vertex e connected to vertices b, d and c via the three edges: (c, e), (b, e) and (d, e).

The following definition justifies our interest in Henneberg construction.

Definition 5 ([21]): A graph G is well-constrained if and only if it has a Henneberg construction.

Proof ([5]): Henneberg construction always generates a well-constrained geometric constraint graph. In reviewing the literature, no study was done on the relation between the choice of Henneberg steps, HI and HII, and the structural propriety of the generated graph. How many solvable sub-graphs have the generated graph if we always chose the first operation HI, or the second operation HII, or if the two operations have the same or different probabilities of being chosen? Let p be the probability of choosing HI. The algorithm is as follows: ■

Algorithm 1. *Generate* a random well-constrained graph using Henneberg construction according to the probability p.

input : n: the size of the graph; p: the probability of choosing HI;
output: a well-constrained graph G.
Procedure *RH* (n, p)
begin
1. start with an initial graph $G = K_3$
2. **for** $i = 1$ **to** n - 3 **do**

3. Generate a random number *r*, uniform on [0, 1);
4. **if** r < p **then** add a new vertex v to G according HI;
5. **else** add a new vertex v to G according to HII;

end if
end for
return (G)
end

Note: in step 3 we use a uniform pseudorandom number generator presented in [18].

To evaluate the degree of decomposability of graphs, generated by the procedure *RH* of algorithm 1, for different values of *p*, three sets of 2000 random graph were generated. Each set corresponds respectively to the value: 0, 0.5 and 1 of the parameter *p*. Due to the randomness, 20 instances for each size ranging from 4 to 100 vertices were generated, and the mean has been calculated.

We have calculated the degree of decomposability using an MM-Planner, based on the Dulmage-Mendelsohn decomposition, presented by Ait-Aoudia et al in [2]. This planner has been chosen because it always returns decomposition if one exists [12].

As it can be seen in Fig. 4(b)., for $p = 0$, which mean that we constantly use the second operation of Henneberg construction HII, generally, graphs of size > 40 were not decomposable. For example, graph of size 20, only 30% were not decomposable, but for size 40 this percentage increase approximately to 80%. In our tests, for the 20 randomly generated graphs, with a size ranging from 40 to 100; rarely there existed a decomposable one. This is due to the use of the second operation of Henneberg, which splits edges. This operation may merge

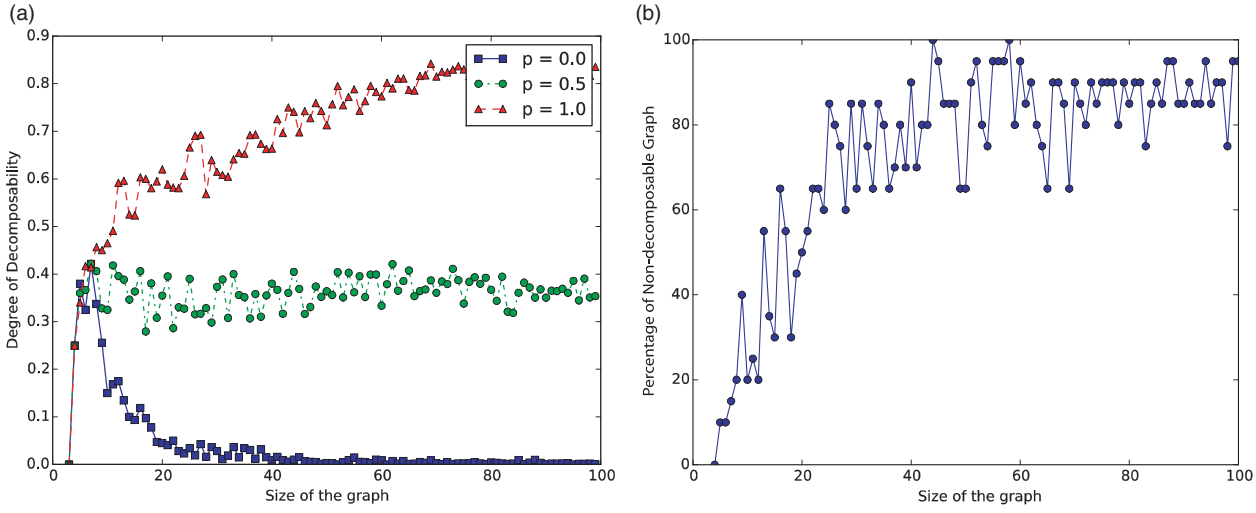

Figure 4. (a) the size of the graph vs degree of decomposability for $p = 0, p = 0.5$ and $p = 1$. (b) The percentage of non-decomposable graphs in each set of 20 instances generated, size ranging from 4 to 100.

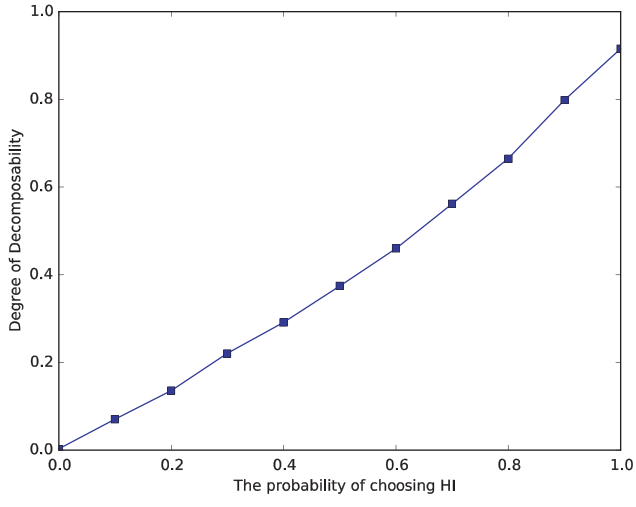

Figure 5. The probability of choosing the first operation of Henneberg vs the degree of decomposition.

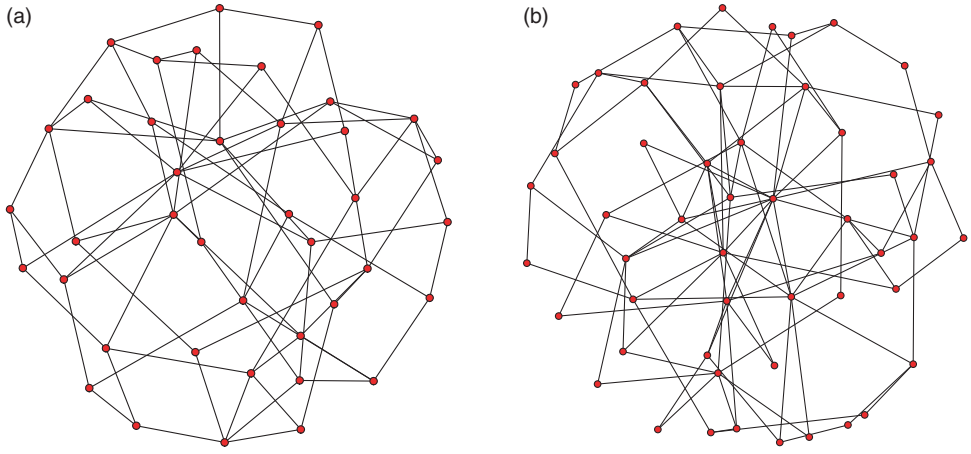

Figure 6. (a) A weakly decomposable graphs generated by *RH*(50, 0). (b) A highly decomposable graph generated by *RH*(50, 1).

Table 1. The degree of decomposition for different values of *p* and size of the graph.

	The probability of choosing HI (*p*)										
Size of graph (*n*)	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
100	0.010	0.062	0.122	0.206	0.276	0.366	0.382	0.576	0.654	0.752	0.862
200	0.005	0.063	0.134	0.184	0.294	0.400	0.456	0.511	0.659	0.787	0.853
300	0.003	0.073	0.121	0.249	0.299	0.351	0.468	0.553	0.617	0.785	0.917
400	0.003	0.074	0.133	0.214	0.299	0.354	0.488	0.547	0.712	0.814	0.911
500	0.002	0.072	0.144	0.234	0.295	0.409	0.483	0.579	0.657	0.825	0.930
600	0.002	0.068	0.145	0.241	0.290	0.367	0.481	0.568	0.641	0.800	0.928
700	0.001	0.072	0.133	0.223	0.309	0.386	0.467	0.584	0.665	0.811	0.939
800	0.001	0.078	0.148	0.231	0.277	0.372	0.475	0.559	0.702	0.793	0.954
900	0.001	0.076	0.144	0.212	0.288	0.359	0.447	0.551	0.653	0.810	0.946
1000	0.001	0.067	0.134	0.211	0.287	0.382	0.454	0.593	0.686	0.811	0.915

two sub-graphs in a single non-decomposable one at each step. This result can be used in the generation of non-decomposable graphs of sizes > 40. A sample non-decomposable graph generated by *RH*, for the case of $p = 0$ and $n = 50$ is given by Fig. 6(a). Fig. 6(b). shows a highly decomposable graph. We note that these two graphs differ in many features. For example, there are many nodes in graph of Figure 6(b). that have a degree of 2, we see the opposite with Figure 6(a).

In the case of $p = 1$, where only the first step of Henneberg construction will be used, whatever the size of the generated graph, it was decomposable. For $p = 0.5$, the degree of decomposition was approximately 0.4. Tab. 1. gives the different values of *d*, for graph size, ranging from 100 to 1000 vertices. We have randomly generated 5 instances for each size and calculated the mean. Results show that the degree of decomposition depends on the probability value *p* rather than on size. In Fig. 5., we have plotted the decomposability degrees for *p* ranging from 0 to 1. It can be seen that the degree of decomposability increases in direct proportion of *p*, i.e. the more we decrease the probability value *p*, the more we weaken the decomposability of the graph. To generate a non-decomposable graphs, *p* must be equal to 0.

Tests conclude that the parameter *p* can be used to parameterize the generator for producing highly or weakly decomposable graphs. In the next section, procedure *RH* of algorithm 1. will be used to generate non-decomposable graphs.

5. Random Constraint Graph Generator

The central role of a Planner is the decomposition of geometric constraints graphs. To evaluate how planners deal, or how will be their behaviour with some types of graphs, a tool for generating situation with some desirable proprieties can be very helpful. Such tool enables the evaluation and the performance analysis of any solving method with more data analysis rather than pure theory.

First, we show how to generate graphs that are decomposable by SR-Planners, and then we explain how to set the size of the larger sub-graph to a desirable value. Decomposability degree is always obtained, using the probability value *p*.

If we want to limit the graph generated by the procedure *RH,* presented in the previous section, to only those that are decomposable by SR-Planners, where the smallest sub-graphs are limited to triangles, the first idea would

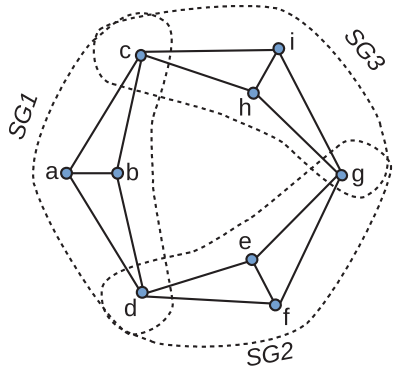

Figure 7. A geometric constraints graph.

be to use only the first operation HI. We will prove that is incorrect, and then we modify the procedure *RH* so that it can easily generate this case using only HI.

Fig. 7 shows a graph that can be decomposed by any SR-Planner. Let shows if there exist a Henneberg construction for this graph that uses only the first operation HI. Supposes that we start with the triangle {*a, b, c*} as the first step, the second step must be no other than adding the vertex *c*. After this stage, no other vertex can be added to the cluster. If we start with another triangle, than {*a, b, c*}, we will obtain the same reasoning because the graph is formed by three symmetric parts: SG_1, SG_2, and SG_3. Hence, this graph is solvable by SR-Planners and not constructible by procedure *RH* using only the first operation HI. Next, we show how to add a recursion mechanism to the procedure *RH* so that any graph decomposable by SR-Planners can be generated using exclusively HI, i.e., where $p = 1$. This will simplify the design of our generator.

We have already seen, that for $p = 0$, graphs of size > 40 generated by procedure *RH* are generally non-decomposable, hence we can produce a larger graph by composition of non-decomposable ones. This procedure can be used to produce a graph that has a predefined size of the largest sub-graph as required by the metric defined in section 2. Generating a graph G is a sequence of graphs $G_1, \ldots, G_n$ with the following properties:

(1) $G_1 = RH(m, p)$;
(2) $G_n = G$;
(3) G_{i+1} is obtained from G_i, through the replacement of random chosen edge of G_i, by a new sub-graph $RH(m, p)$.

Let m be the desired size of the largest sub-graph, the algorithm follows.

Algorithm 2. Generate a **random** well-constrained graph, according to the two metrics:

(1) the size of the largest sub-graph
(2) the degree of decomposition.

Input : *n*: size of the graph;
p: probability of choosing the first step of Henneberg construction;
m: the size of the largest sub-graph;
Output: *G*: a well-constrained graph
Procedure *RRH (n, p, m)*
begin
1. start with a graph $G = K_3$
2. $k = n / m$
3. **for** $i = 1$ **to** k **do** / * we generate k sub-graph, each one have a size m * /
4. generate a random graph H of size m using *RH(m, p)*;
5. pick an edge (x, y) from G
6. replace the edge (x, y) by the graph H as follows:
7. pick two random vertices *x'* and *y'* from the graph H
8. connect *x'*, to all neighbours of x
9. connect *y'*, to all neighbours of y
10. delete x and y

endfor
11. Complete G by adding a sub-graph $H = RH(n - |V| + 2, p)$ as in step 4–10.
12. return G

end

Notice, as seen in section 3, generally the procedure *RH* generates a non-decomposable graph for $m > 40$ and $p = 0$. For $m < 40$, mostly, procedure *RRH* will be not efficient. In this case, instead of generating non-decomposable graphs using *RH*(*m, p*), we can easily design a dataset of non-decomposable graphs, having a size lower than 40. Step 4 of the procedure *RRH* will be replaced by a random retrieval from this dataset. If we set the parameter m to a desirable value of size of the largest sub-graph, the parameter p must be equal to zero.

Claim 1: Any structurally well-constrained graph can be generated by *RRH*.

Proof: Since *RRH* uses Henneberg construction, and any one of the two operations (HI or HII) may be executed in each step of *RRH*, and by the definition 5., we conclude that *RRH* generates all the domain of the well-constrained graph in 2D. ■

Claim 2: A geometric constraints graph is solvable by any SR-Planner if and only if $p = 1$ or $m = 3$.

Proof: If $m = 3$ then the procedure *RRH* will start initially by a triangle, and recursively replace edges by new triangles. If $p = 1$, i.e., *RRH* uses only the first operation of Henneberg: H1, then *RRH* start with a triangle

and recursively add new vertex and connecting it to two vertices of the graph.

Replacing an edge of a triangle by a triangle, or connecting a new vertex to two vertices of a triangle lead to the same graph.

We recall that all SR-planners solve only a sub-domain of GCS. in [14], Joan-Arinyo et al. studied the domain of SR-planners, they have proved in that Owen's SR-planner proposed in [19] and the planner proposed by Fudos et al. in [4] solve the same domain of GCS,. Recently, he proposed in [15] a new SR-Planner that also solves the same domain.

To prove the claim 2, we have to prove that if a graph G is solvable by an SR-Planner (called reduction analysis) proposed by Fudos et al. in [4], then, it can be generated by procedure RRH for $p = 1$.

Let us recall now the principal of the reduction analysis algorithm. More details can be found in [4].

Given the constraint graph $G = (V,E)$, we consider the initial set of clusters $C_G = \{\{u,v\} / (u, v) \in E\}$. Cluster sets, that have a central role in this method, are rewritten using a reduction $\rightarrow$. The reduction $\rightarrow$ is formally defined as follows: Let S be a set of clusters in which there are three clusters C_1, C_2, and C_3 such that: $\exists\, g_1, g_2, g_3 \in V$, $C_1 \cap C_2 = \{g_1\}$; $C_1 \cap C_2 = \{g_2\}$; $C_1 \cap C_2 = \{g_3\}$ and $g_1 \neq g_2 \neq g_3$. (i.e., those tree clusters pairwise intersection in a singleton). Then: $S \rightarrow S'$, where $S' = S - \{C_1, C_2, C_3\} \cup \{C_1 \cup C_2 \cup C_3\}$.

The solving process is a repetitive application of reductions $\rightarrow$, that start from the initial cluster S_G. If this process end with a singleton $\{V\}$, i.e. the final cluster contain only on element which is the set of nodes V, then the graph is solvable by the reduction analysis algorithm.

First, we prove by induction that if $p = 1$, then the generated graph is solvable by the reduction analysis algorithm.

Because $p = 1$, the procedure *RRH* always uses the first operation HI (in step 4). The generation process of the graph G by *RRH* a sequence of graphs $G_1, \ldots, G_n$ with the following properties : $G_1 = K_3$; $G_n = G$; and G_{i+1} is obtained from G_i , though only the operation HI.

1. G_1 is a triangle; hence it is decomposable by the reduction analysis algorithm (trivial).
2. Suppose that G_i is decomposable by SR-Planners. G_{i+1} is obtained by adding a new vertex v to G_i and connecting it to two randomly chosen vertices of G_i: u and w. Because G_i is supposed decomposable by the reduction analysis algorithm, then it corresponding set of clusters can be reduced to a single set SG_i where $u,w \in SG_i$. $SG_i \cap \{u,v\} = \{u\}$. $SG_i \cap \{w,v\} = w$ and $\{u,v\} \cap \{w,v\} = v$. Hence, the set of clusters: $\{SG_i, \{u,v\}, \{w,v\}\} \rightarrow \{SG_i \cup \{u,v\} \cup \{w,v\}\} = SG_{i+1}$. We deduce that G_{i+1} is Decomposable by SR-Planners.

Now we prove that *RRH* generate all the domain of SR-Planners if $p = 1$ *or* $m = 3$.

Let G be a graph that is decomposable by the reduction analysis algorithm. Then there exist a cluster reduction: $SG_1 \rightarrow SG_2 \rightarrow \ldots SG_{k-1} \rightarrow SG_k$. We prove that there is a sequence of steps in the procedure RRH that generate G.

S_{K-1} is the penultimate set of clusters of the reduction analysis, then it has three elements = {L, M, N}, where $L \cap M = \{g_1\}$; $L \cap N = \{g_2\}$; $M \cap N = \{g_3\}$. The last step of the reduction corresponds to the first step of the procedure *RRH*. (step 1 in algorithm 2). It creates a triangle, let be $\{g_1, g_2, g_3\}$. The three edges of this triangle: (g_1, g_2), (g_1, g_3), (g_2, g_3), will be replaced respectively (steps 4 and 5 of algorithm 2) by three graphs: G_L, G_M and G_N that corresponds to the three clusters: L, M and N. If a cluster C has only two elements, i.e., it corresponds to an edge, this edge will not be replaced by any graph, but considered as a graph composed by only on edge.

Because the three graphs G_L, G_M and G_N are well-constrained, there exist a reduction sequence for every one of them, and the last reduction of each sequence has a corresponding step in the procedure RRH that can be demonstrated in the same manner as the last reduction sequence of G shown above.

Because there is a finite number of reduction, we deduce that, if a graph has a reduction sequence, i.e. it is solvable by any SR-Planner, then it can be generated by the procedure RRH for $p = 1$. ■

6. Experimental Results

We conducted experiments to evaluate the solvability of graphs generated by procedure *RRH*. Each generated graph that have a known number of sub-graphs will be decomposed to show how those sub-graphs are detected. Experiments are done for graphs size n in {500,600,700,800,900,1000} and for the largest sub-graph size m in {50,10,150,200,250,300,350,400,450,500}. To ensure that sub-graphs are not decomposable, we fixed the parameter p to *0* and the size m of the smallest sub-graph to 50. We recall that for the others values of p and m, experiments are presented in section 3.

We calculate the mean of the number of sub-graphs detected, after decomposition. Due to randomness, for each size, we generate 20 instances of each situation and calculate the mean. Tab. 2 gives the average number of sub-graphs detected after decomposition.

Theoretically, we expect that the number of sub-graphs detected is at least equal to n / m. Tab. 2. shows the

Table 2. The average number of sub-graph detected for graphs generated by *RRH* (*n, 0, m)* for different values of *n* and *m*.

	Size of the largest sub-graph *(m)*									
Size of the graph *(n)*	50	100	150	200	250	300	350	400	450	500
500	5.850	3.750	3.000	2.850	2.300	2.050	1.450	1.400	1.700	1.100
600	7.300	3.550	2.450	2.950	2.600	2.050	2.400	2.100	1.050	1.200
700	5.800	3.350	3.250	2.950	2.800	2.450	2.150	2.000	2.000	2.000
800	7.800	4.500	3.400	2.900	3.100	2.600	2.600	2.050	2.250	2.050
900	7.500	3.500	3.600	3.000	2.600	2.800	2.650	2.650	2.000	2.000
1000	8.600	3.550	3.650	3.250	2.700	2.900	2.600	2.450	2.700	2.200

opposite. For example, for $n = 500$ and $m = 50$, instead of detecting at least $500 / 50 = 10$ sub-graphs, the decomposition returns only an average of 5.85. The method of decomposition that we used did not really detect all sub-graphs, nor did other decomposition methods. This is explained by the fact that finding sub-graph of minimal size is NP-Hard as reported by Hoffmann et al. in [11]. We note that For *RRH* (500,0,500), the average was 1.1, as we were expecting. Notice, those results may change with others decomposition algorithm. Fig. 8 shows a large graph composed of 500 vertices, generated by *RRH*(500,0,50). It contains 10 non-reducible sub-graphs, each one is generated in step 4 of algorithm 2 and has a size of 50 nodes. To be differentiated, sub-graphs are plotted in different colours. Some sub-graphs will be merged because the creation process is a recursive replacement of edge of the graph by a new generated sub-graph H (step 6 of algorithm 2.). In Fig. 8. The yellow sub-graph is merged with the orange and the brown with light-blue. To find the best decomposition possible, a good planner must be able to isolate all of them. In Tab 2. (The top-left cell of the table), the average number of detected sub-graphs is 5.85 but the real number is $500 / 10 = 10$.

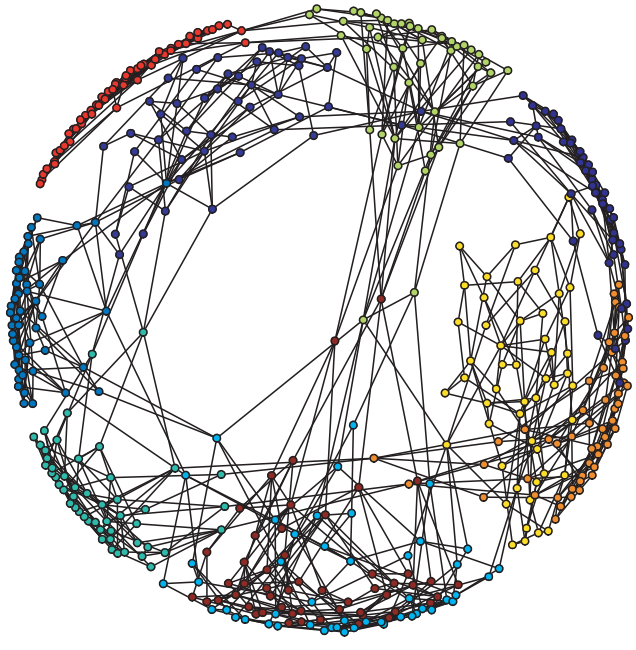

Figure 8. A well-constrained graph of 500 vertices generated by *RRH(500,0,50)*.

7. Conclusion

We presented two algorithms for generating 2D geometric constraint graphs. The first one, *RH*, can be used to produce non-decomposable graphs or graphs with a given degree of decomposability. The second algorithm, *RRH*, can serve as a generator of graphs with desired size of the larger sub-graph. It can also be parameterized to generate graphs that are solvable by SR-Planners or MM-planners. We conducted an experimental study to show how generated graphs are decomposable. Our graph generator is complete: generate all the domain of well-constrained geometric systems; Fast: linear on the number of iteration; Customizable: it requires few parameters to generate a class-specific graph with desired properties. It can be used to test and observe the behaviour of many SR-planners or MM-planners, moreover, it enables the comparison of solving methods with more data analysis rather than pure theory. Efficient and simple: based on strong theorems and simple to be implemented. It has been validated experimentally by decomposing generated graphs with a well-chosen planner.

Acknowledgements

We thank two anonymous reviewers for their careful reading of the paper, their suggestions and comments.

ORCID

Adel Moussaoui http://orcid.org/0000-0002-6978-1656
Samy Ait-Aoudia http://orcid.org/0000-0002-6074-2060

References

[1] Ait-Aoudia, S.; Foufou, S.: A 2D geometric constraint solver using a graph reduction method, *Advances in Engineering Software*, 41(10), 2010, 1187–1194. http://dx.doi.org/10.1016/j.advengsoft.2010.07.008

[2] Ait-Aoudia, S.; Jegou, R.; Michelucci D.: Reduction of constraint systems, *Compugraphics'93 Alvor, Algarve, Portugal*, 1993, 331–340.

[3] Bettig, B.; Hoffmann C. M.: Geometric constraint solving in parametric computer-aided design, *Journal of Computing and Information Science in Engineering*, 11(2), 2011, 021001. http://dx.doi.org/10.1115/1.3593408

[4] Fudos, I.; Hoffmann, C. M.: A graph-constructive approach to solving systems of geometric constraints, *ACM Transactions on Graphics (TOG)*, 16(2), 1997, 179–216. http://dx.doi.org/10.1145/248210.248223

[5] Haas, R.; Orden, D.; Rote, G.; Santos, F.; Servatius, B.; Servatius, H.; Souvaine D.; Streinu I.; Whiteley, W.: Planar minimally rigid graphs and pseudo-triangulations, Proceedings of the nineteenth annual symposium on Computational geometry, *ACM*, 2003, 154–163. http://dx.doi.org/10.1016/j.comgeo.2004.07.003

[6] Hendrickson, B.: Conditions for unique graph realizations, *SIAM Journal on Computing*, 21(1), 1992, 65–84. http://dx.doi.org/10.1137/0221008

[7] Henneberg L.: *Die graphische Statik der starren Systeme*, Leipzig, 1911, Johnson Reprint, 1968.

[8] Hoffmann, C. M.; Joan-Arinyo R.: A brief on constraint solving, *Computer-Aided Design and Applications*, 2(5), 2005, 655–663. http://dx.doi.org/10.1080/16864360.2005.10738330

[9] Hoffmann, C. M.; Lomonosov, A.; Sitharam, M.: Decomposition plans for geometric constraint systems, Part I: Performance measures for CAD, *Journal of Symbolic Computation*, 31(4), 2001, 367–408. http://dx.doi.org/10.1006/jsco.2000.0402

[10] Hoffmann, C. M.; Lomonosov, A.; Sitharam, M.: Decomponosition plans for geometric constraint problems, part II: new algorithms, *Journal of Symbolic Computation*, 31(4), 2001, 409–427. http://dx.doi.org/10.1006/jsco.2000.0403

[11] Hoffmann, C. M.; Lomonosov, A.; Sitharam, M.: Geometric constraint decomposition. In Geometric constraint solving and applications, *Springer Berlin Heidelberg*, 1998, 170–195. http://dx.doi.org/10.1007/978-3-642-58898-3_9

[12] Jermann, C.; Trombettoni, G.; Neveu, B.; Mathis, P.: Decomposition of geometric constraint systems: a survey, *International Journal of Computational Geometry & Applications*, 16(05n06), 2006, 379–414. http://dx.doi.org/10.1142/S0218195906002105

[13] Joan-Arinyo, R.; Soto-Riera, A.; Vila-Marta, S.; Vilaplana-Pasto, J.: Revisiting decomposition analysis of geometric constraint graphs, *Computer-Aided Design*, 36(2), 2004, 123–140. http://dx.doi.org/10.1016/S0010-4485(03)00057-5

[14] Joan-Arinyo, R.; Soto-Riera, A.; Vila-Marta, S.; Vilaplana, J.: On the domain of constructive geometric constraint solving techniques, in Computer Graphics, Spring Conference on Computer Graphics, *IEEE*, 2001, 49–54. http://dx.doi.org/10.1109/SCCG.2001.945336

[15] Joan-Arinyo, R.; Tarrés-Puertas, M.; Vila-Marta, S. : Decomposition of geometric constraint graphs based on computing fundamental circuits. Correctness and complexity, *Computer-Aided Design*, 52, 2014, 1–16. http://dx.doi.org/10.1016/j.cad.2014.02.006

[16] Laman, G.: On graphs and rigidity of plane skeletal structures, *Journal of Engineering mathematics*, 4(4), 1970, 331–340. http://dx.doi.org/10.1007/BF01534980

[17] Latham, R.; Middleditch, A.: Connectivity analysis: a tool for processing geometric constraints, *Computer Aided Design*, 28(11), 1996, 917–928. http://dx.doi.org/10.1016/0010-4485(96)00023-1

[18] Matsumoto M.; Nishimura T.: Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator, *ACM Transactions on Modeling and Computer Simulation*, 8(1), 1998, 3–30. http://dx.doi.org/10.1145/272991.272995

[19] Owen, J. C.: Algebraic solution for geometry from dimensional constraints, in Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications, *ACM*, 1991, 397–407. http://dx.doi.org/10.1145/112515.112573

[20] Saxe J.: Embeddability of weighted graphs in k-space is strongly NP-hard, Proceedings of 17th Allerton Conference in Communications, *Control and Computing*, 1979, 480–489.

[21] Tay T.S.; Whiteley W.: Generating isostatic frameworks, *Structural Topology*, 11, 1985, 21–69.

[22] Zhang, Y.; Liu, S.; Zhao, X.; Jia, Z.: Theoretic analysis of unique localization for wireless sensor networks, *Ad Hoc Networks*, 10(3), 2012, 623–634. http://dx.doi.org/10.1016/j.adhoc.2011.06.016