Graph Constructive Geometric Constraint Solving: Challenges and Machine Learning

Ioannis Fudos[1] and Vasiliki Stamati[2]

[1]University of Ioannina, fudos@uoi.gr
[2]University of Ioannina, vstamati@uoi.gr

Corresponding author: Ioannis Fudos, fudos@uoi.gr

Abstract. Computer-aided design (CAD) places significant emphasis on crafting precise and durable models that seamlessly adhere to constraints set forth by designers and/or specific application domains. Geometric constraint solving (GCS) plays a pivotal role in ensuring the fulfillment of these criteria. This paper delves into the contemporary research problems encountered within traditional GCS methodologies, particularly when combining graph algorithms (termed graph constructive GCS) and machine learning. Specifically, it investigates challenges about well-constrainedness in both 2D and 3D GCS scenarios. Furthermore, it proposes and scrutinizes an AI-assisted root navigation approach for graph-based constructive constraint-solving problems.

Keywords: Geometric constraint solving, CAD, root navigation, animation, robotics, rigidity, machine learning
DOI: https://doi.org/10.14733/cadaps.2025.995-1006

1 INTRODUCTION

Computer-aided design (CAD) has played a crucial role in various engineering applications such as modeling, analysis, optimization, manufacturing, and maintenance. The concept of CAD dates back to the 1960s, but it was the introduction of parametric CAD modelers in the late 1980s that led to widespread recognition within the user and engineering community.

Parametric CAD allows for geometry reuse, enabling users to create a family of geometry variants by adjusting embedded parameters in the model. Parametric modeling software is used in various industries such as industrial machinery, consumer products, automotive and aerospace, and healthcare. It helps in creating detailed 3D models, optimizing designs, simulating operations, and generating manufacturing-ready designs. The associativity of parametric CAD techniques allows for automatic change propagation when local parametric changes are made, achieved through a system of geometric constraints and constraint solving.

Feature-based computer-aided design (CAD) is a design methodology that uses predefined shapes or features to create complex 3D models. These features can be simple shapes like lines, arcs, and circles or more complex shapes like extrusions, chamfers, and fillets. Feature-based CAD allows designers to create models more quickly and easily than with traditional CAD methods, which require users to manually draw each line and shape. Features can be easily modified and updated, making it easier to make changes to a design. Feature-based CAD is also more parametric than traditional CAD, meaning that the model can be easily updated if the underlying parameters are changed. However, since feature-based CAD involves design parts arbitrarily interconnected through various geometric constraints it needs to be supported by a powerful, robust, and efficient geometric constraint-solving engine. Overall, feature-based CAD is a powerful tool that can help designers create complex 3D models quickly and easily.

Geometric constraint solving (GCS) in Computer-Aided Design ensures that the elements of a design maintain their intended relationships and properties, thereby facilitating the creation of accurate, precise, and consistent parts. Additionally, GCS enables the automation of design modifications, leading to a more efficient use of time and resources. Moreover, geometric constraints help enforce standardization in design practices, ensuring that designs adhere to specific rules and guidelines.

Geometric constraint solving involves the use of a geometric constraint system consisting of geometric entities and constraints. The process of solving systems of geometric constraints involves translating constraints into algebraic equations and finding valid instantiations of geometric entities that satisfy all the constraints. Challenges in GCS include characterizing constraint states, detecting ill-constrained parts, and decomposing geometric constraint systems into smaller subsystems for efficient solving. Research efforts have focused on developing effective criteria for detection and decomposition algorithms for general constraint systems. Various techniques have been employed for GCS, including rule-based constraint solving, nonlinear optimization using homotopy, analysis of the graph of geometric relationships, and constructive constraint solving.

In this paper, we explore graph constructive constraint solving by identifying the major challenges and suggesting plausible research directions that leverage machine learning and theoretical breakthroughs.

More specifically, we investigate the use of graph neural networks to encode a geometric constraint solving problem as a matter of relationships among elementary rigid bodies. Each node in the graph represents a rigid body, and methods are developed to detect whether a node belongs in a minimal constructible rigid body cluster. This approach provides a sequence of rigid body placements that will derive a solution for the GCS problem, making it explainable, user-intuitive, and aligned with design intent. Additionally, we address the root selection or root navigation problem by employing a neural network to select the appropriate root (solution) for an elementary construction, trained on examples drawn from standard design libraries.

We will also delve into the theoretical foundations for GCS and attempt to provide novel, sufficient, and necessary conditions for the well-constrainedness of 2D and 3D GCS problems.

The deep graph constructive GCS approach suggested may have applications in various fields besides CAD, including kinematics analysis for robotics, determining the degrees of freedom in animation sequences, molecular biology for determining feasible placements of molecules in 3D, and other applications that involve geometric constraints in both 2D and 3D spaces.

2 STATE OF THE ART

Numerical methods offer powerful tools for iteratively solving large systems of equations. Typically, ensuring convergence requires providing a reasonably accurate initial approximation of the desired solution. Therefore, if the initial point is derived from the user-defined sketch, it should closely align with the intended solution. The widely adopted Newton-Raphson stands out as a local method used in solvers described by various sources (see, e.g., [10]). Homotopy or continuation [5], [14], [16] represents a family of global methods that can determine all solutions but exhibit lower efficiency compared to local methods.

Symbolic algebraic techniques calculate a Grobner basis for a given system of equations with notable algorithms (see, e.g. [19]). Analytical methods focusing on system structures assess whether a system is under-constrained, well-constrained, or over-constrained. These approaches can be extended to decompose systems into minimal graphs, solvable independently [18], serving as a pre-processing step to reduce the variables and equations to be solved simultaneously. These methods can be applied to almost any system of equations but are slow (sometimes require exponential time on the number of equations), cannot capture over and under-constrained systems, and do not provide an intuitive placement of geometric elements according to geometric constraints since they do not exploit the geometric features of the problem.

In logic-based approaches (aka rule-based), the problem undergoes translation into a set of assertions and axioms that characterize both the constraints and the geometric objects. Researchers (see e.g. [4]) employ first-order logic to extract geometric information using a set of axioms derived from Hilbert's geometry. These methods essentially generate geometric loci representing the locations where the elements must be positioned. [13] extend these sets of constraints. These methods are nice for prototyping and testing new repertoires of constraints, geometries, and rules but cannot be used in real-world systems since they use exhaustive matching techniques. However, recent advances in machine learning for geometric algebra may have interesting implications (see e.g. [3]).

The constructive approach entails deriving solutions to geometric constraint problems through a sequence of elementary construction steps. Each step follows rules from a predetermined set of operations, positioning specific geometric elements. This method effectively preserves the inherent geometric meaning of each operation in the solution. Depending on the analytical technique applied, constructive approaches can be categorized as either top-down or bottom-up (see [12] for an overview). [8],[9] use clusters as rigid body sets of geometries and constraints, employing a rule that merges three clusters that, pairwise, share one element. This approach provides a unified approach to treat geometric constraint systems as rigid bodies where constraints are reduced to incidence constraints (rigid bodies that share a geometry). While this line of work provides a fast and intuitive approach to the GCS problem three major issues should be addressed [1], [12]: (i) extend the repertoire of rigid body patterns that can be merged/placed and (ii) extend the repertoire of elementary geometric object to include higher dimension geometry (e.g. circle with variable radius) (iii) extend the algorithm to 3D.

Degrees of Freedom (dof) Analysis involves assigning degrees of freedom to geometric elements by labeling the vertices of the constraint graph. Each graph edge is tagged with the number of degrees of freedom eliminated by the associated constraint. The method then resolves the problem by examining the resulting labeled graph. [8] uses reduction systems to detect triangle decomposable well-constrained systems. [14] breaks down the labeled graph into minimal connected components called balanced sets. [11] presents a flow-based method for decomposing graphs of geometric constraint problems. The method iterates to decompose the underlying algebraic system into minimally dense subgraphs. While this method fully generalizes degree-of-freedom calculations and prior flow-based approaches, the resulting decomposition may lack geometric sense, as minimal dense subgraphs can exhibit arbitrary complexity beyond classical geometric construction problems. In 2D, the Laman theorem provides a sufficient and necessary condition for characterizing the well-contrainedness of a GCS if the geometries have two degrees of freedom and the geometric constraints reserve one degree of freedom. If we include geometries with three degrees of freedom the Laman condition is not sufficient anymore. A necessary and sufficient condition for rigidity in 3D has been presented in [15] for an extended set of geometries and constraints. Note that this characterization does not capture the cases explained in Figures 4 and 5. The condition holds for both cases, but rigidity is not guaranteed since they involve point coincidences between rigid bodies directly or indirectly. Leveraging a similar analysis on the system of equations that describe a GCS

[7],[17] has presented the witness configuration method, which also has many limitations [20] and cannot be applied to solve large GCS problems.

Overall, while the geometric constraint-solving problem has been extensively addressed from various perspectives, each approach has its limitations and strengths. For instance, numerical methods can solve large nonlinear systems but are highly dependent on the initial approximation. Logic-based methods consider geometric properties and are efficient for small-scale problems, but their real-world applicability is limited. Constructive approaches more effectively exploit geometric features, capture design intent, and hold potential for further research. With this in mind, we delve into the realm of graph-constructive GCS and explore the prospective benefits of integrating AI methodologies.

3 CHALLENGES AND RESEARCH DIRECTIONS

Considering the benefits of GCS and the above analysis of methods developed and applied in this field, the following major challenges can be identified both generally in CAD and graphics and, more specifically, in reference to GCS.

(C1) Making CAD and animation tools easier to use and more productive involves improving the user interface, streamlining workflows, and enhancing the functionality of the software. This could include features such as intuitive controls, real-time feedback, and simplified processes for creating and editing designs or animations. The goal is to make the tools more accessible and efficient for users, ultimately increasing their productivity.

(C2) Advancing theoretical foundations for geometric constraint solving entails further developing the mathematical principles and algorithms that underpin the process of solving geometric constraints. This could involve research into new mathematical models, optimization techniques, and computational methods to improve the accuracy, efficiency, and robustness of geometric constraint solving.

(C3) Building powerful and explainable geometric constraint-solving engines involves creating software systems that are capable of accurately and reliably solving complex geometric constraints while providing clear explanations for their solutions. This could include developing algorithms that can handle a wide range of geometric scenarios, providing detailed reasoning for each step of the solving process, and ensuring transparency in the decision-making of the solving engine.

We propose the following directions of research on graph constructive GCS and explain how they affect the above challenges:

(O1) Enhance the theoretical foundations by (i) comprehensively understanding the structural properties that ensure well-constrainedness (addressing challenges C2 and C1), and (ii) proposing methods for effectively solving consistently over and under-constrained systems (addressing challenge C3).

(O2) Develop an efficient and versatile technique for deriving a construction sequence for placing geometric elements that satisfy the imposed geometric constraints (addressing challenge C1). The construction sequence should be easy to explain and align with design intent (addressing challenge C3).

(O3) Introduce a user-friendly root selection process, enabling users, designers, and artists to navigate through the solution space. The process will be powered by a deep learning method that automatically recommends the most plausible root based on supervised learning from examples drawn from CAD libraries (addressing challenges C1 and C3).

(O4) Utilize advanced deep graph constructive GCS techniques for (i) editing parametric and feature-based CAD models, (ii) generating novel feature-based CAD models quickly and with minimal user intervention, (iii) constraining and animating articulated characters, and (iv) animating machine parts and robotic setups by deriving the degrees of freedom of the geometric elements (all addressing challenge C1).

4 MACHINE LEARNING FOR GRAPH CONSTRUCTIVE GCS

Definition 1: Let P be a geometric constraint system *P = (U, F)* where *U* are the geometric elements and *F* are the constraints imposed on them. The constraint graph *G = (V, E)* of *P* is a labeled undirected graph whose vertices are the geometric objects in *U*, each labeled with its degrees of freedom. There is an edge *(u, v)* in *E* if there is a constraint between the geometric objects *ui* and *vi,* corresponding to *u* and *v,* respectively. The edge is labeled by the number of independent equations corresponding to the constraint between *ui* and *vi.*

Below, we will omit the edge labels when they represent one degree of freedom and the labels of the vertices when they are 2 in 2D and 3 in 3D. Every geometric element has a certain number of degrees of freedom. For example, a line in 3D has four degrees of freedom. Each constraint reserves one or more degrees of freedom. For example, in 3D, a coincidence constraint "point on line" reserves one degree of freedom, whereas a coincidence constraint "point on point" reserves three degrees of freedom. The deficit of a constraint graph is the sum of degrees of freedom minus the reserved by constraints degrees of freedom.

Tables 1 and 2 are adopted from [12] and show the dof of the elementary geometric objects and the dof that are consumed by several types of geometric constraints. For geometric constraints, each simple (non-vector) equation is considered to cancel exactly one degree of freedom.

Geom	Geometric Meaning of dof	2D	3D
Point	Variables representing coordinates	2	3
Line	2D: distance from origin and direction 3D: distance from origin, direction in 3D direction on the plane	2	4
Plane	Distance from origin, direction in 3D		3
Circle, fixed-radius	Coordinates of center, orientation in 3D	2	5
Circle, variable-radius	Coordinates of center, radius, orientation in 3D	3	6
Rigid body	2D: 2 displacements, 1 orientation 3D: 3 displacements, 3 orientation	3	6
Sphere, fixed- or variable-radius	3 displacements or 3 displacements, radius	3	4
Ellipse, variable axes	center, axis lengths, axis orientation	5	7
Ellipsoid, variable axes	center, axis lengths, axis orientation		9

Table 1: Degrees of freedom for elementary objects and rigid objects.

Type	Constraint	2D	3D
point-point distance	One equation representing the distance between two points p1, p2 under the metric $\|\| \|\|$: $\|\|p1 - p2\|\| = d$ with $d > 0$.	1	1

Angle between lines and planes	Angle between two lines (can be represented by the angle between the normal vectors). Exceptions in 3D: Parallelism between lines eliminates 2 dof. Line-plane orthogonality in eliminates 2 dof.	1	1
Point on point	For any metric ||p1 − p2|| = 0 is equivalent to p1 = p2.	2	3
Point-line distance	One equation expresses the point-line distance.	1	1
Line-line parallel distance	Parallelism and distance.	2	3
Plane-plane parallel distance	Parallelism and distance.	2	3
Point on line	Same as point-line distance In 3D dimension is reduced.	1	2
Line on line	Same as parallel distance between lines in 2D. In 3D an additional dof is canceled.	2	4
Point on plane	Same as point-plane distance.	-	1
Line on plane	Plane-line parallelism and zero distance.	-	2
Fixing elementary object	Fixing all or some of the dof of an elementary object.	dof	dof

Table 2: Types of constraints and number of dof eliminated.

Figure 3 shows a geometric constraint problem in 2D which cannot be solved by merely placing objects sequentially. If we create the three rigid bodies as shown in the middle (from A, I we place H, then from I and H we place G and so on) we will end up with the situation shown on the right. There, we must place points A, D and G and then rotate and translate the three rigid bodies to fit on the three points.

This implies that the geometric constraint system can be solved by successive reductions of well-constrained rigid body configurations. Rigid bodies share geometric elements. Shared geometric elements can be represented as coincidence constraints that eliminate the same number of degrees of freedom as those of shared geometry. For example, each shared point in 2D eliminates two degrees of freedom. This is a powerful and intuitive GCS method that can be applied in 2D or 3D. Currently, there are certain limitations that we will attempt to overcome in this project.

A more complex example of a geometric constraint problem is presented in Figures 1 and 2. This example involves two Bezier curves that are used for creating a fillet that blends the two sides of a box when extruded or swept to 3D. The constraint problem is compiled into a problem with lines, points, distances, angles, and constraints with 12 geometries (with 24 DOF) and 21 edges that each consumes one of, deriving a rigid body in 2D with 3 DOF. This gives a well-constrained problem that is sequentially constructible. Here is a construction sequence: P_A, P_B are placed at distance d_1, l_4 is constructed from P_A and P_B, l_3 is placed from P_B and l_4, l_2 is constructed from P_A and l_4, P_1 from l_2 and P_A, l_5 from P_1 and l_2, P_5 form l_5 and l_3, P_4 from P_5 and l_3, P_2 from P_1 and l_2, l_1 from P_2 and P_4, and finally, P_3 is constructed from l_1 and P_2.

4.1 Theoretical Foundations: Extending Laman's Theorem

Laman's theorem determines whether a geometric constraint problem is well constrained, meaning that the problem has a finite number of solution instances for non-degenerate configurations.

Laman theorem: Let *G = (V, E)* be a connected, undirected graph whose vertices represent points in 2D and edges represent distances between points. *G* is a well-constrained constraint graph of a geometric constraint problem iff, the deficit of G is 3, and, for every subset *U ⊂ V,* the induced subgraph *(U, F)* has a deficit of no less than 3.

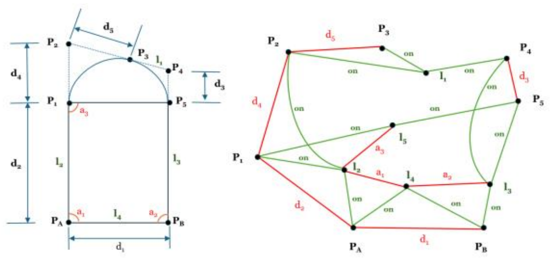

Figure 1: A geometric constraint problem (left) illustrating the profile of a box with two blending Bezier surfaces. For the 2D profile, we have two conic Bezier curves. The 2D problem translates to the constraint graph in the (right) with 12 elementary geometric objects in 2D and 21 constraints.

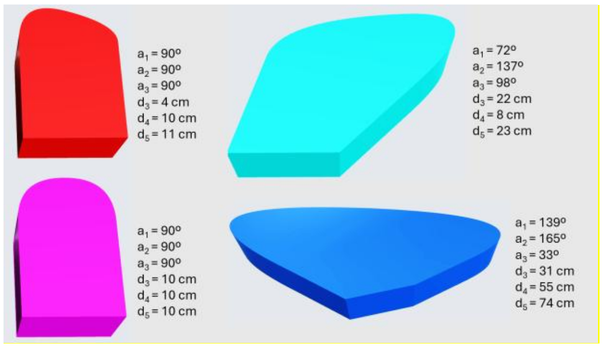

Figure 2: 3D objects that result from different constraint values of the profile of Figure 1. In all configurations d1= 20 cm and d2 = 20 cm

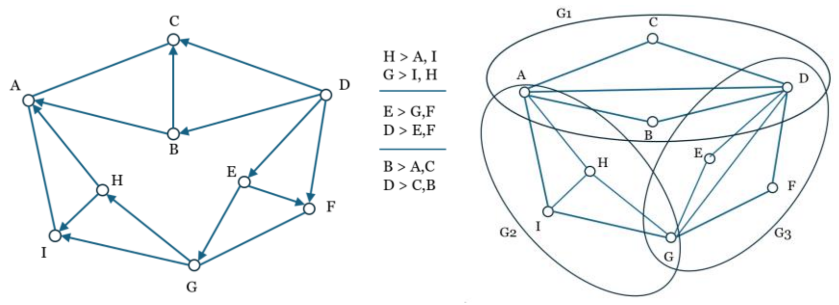

Figure 3: (Left) a GCS problem (A through I are points in 2D and edges represent distances), the constraint graph is the same undirected graph. (Middle) three plausible construction sequences for placing the three groups of points, every group of points represents a rigid body. (Right) Three rigid bodies share a point pairwise.

Laman's theorem holds even if we extend the repertoire of geometries to any geometry having 2 degrees of freedom and the constraints to virtually any constraint. However, if we extend the set of geometries to include, for example, variable radius circles (which acquire 3 degrees of freedom), then the Laman condition is no longer sufficient.

Figure 4 depicts a Laman graph in 2D that is not rigid. V_1 is a variable radius circle. V_2, V_3, V_2', V_3' are points. V_4, V_4' are lines. e_1, e_2, e_1', e_2' are point-to-circle distances (distances from the center of the circle). e_3, e_4, e_3', e_4' are on constraints. e_5, e_5' are distances. e_6, e_6' are distances from the circle (circumference).

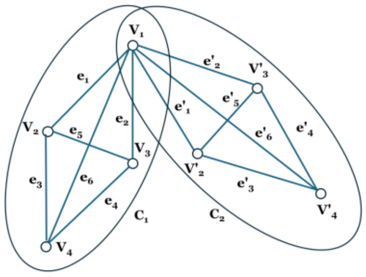

Figure 4: A constraint graph in 2D. It consists of two rigid bodies that share a 3 dof geometry. Although the Laman condition is valid, the graph is not well-constrained.

Graph analysis for spatial constraint problems is not nearly as mature as the planar case. In 3D, the Laman condition is not sufficient. Figure 5 illustrates two hexahedra sharing two vertices. If the length of the edges is given the GCS that arises is also known as the double banana problem. The graph is a Laman graph but the problem corresponds to two rigid bodies (each hexahedron is a rigid body) sharing two vertices and is thus non rigid in the sense that the two rigid bodies are free to rotate around the axis defined by the two shared points. The problem is also clearly overconstrained since the distance of the two shared geometries can be derived independently by each of the two rigid bodies.

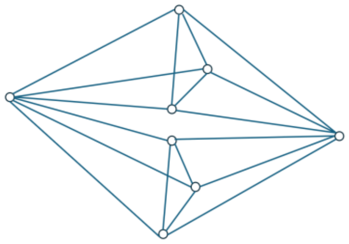

Figure 5: A constraint graph in 3D. Also known as the double banana problem, since it reduces to two rigid bodies that share two points. Although the Laman condition is valid, the graph is not well-constrained.

Therefore, we propose the following conjecture that extends Laman's theorem:

Conjecture 1: A geometric constraint problem is well constrained iff the reduction sequence does not derive two rigid bodies that share geometries with a total degree of freedom equal to the degree of freedom of a rigid body in the corresponding space (6 for 3D, 3 for 2D).

If such rigid bodies (that share pairwise 3 in 2D or 6 in 3D dof) appear we can safely conclude that the problem is not well constrained. It remains to determine whether the reverse holds. We shall study the correctness of Conjecture 1 and provide an efficient algorithm using deep graph reduction to determine the well-constrainedness of a geometric constraint scheme.

Extending the repertoire of reductions: Deep graph reduction

The simplest reduction is the triangle reduction (see Figure 3 (right)) which has been studied extensively. Figure 6 (left) illustrates two more minimal cases of well-constrained configurations of graph constraint problems.

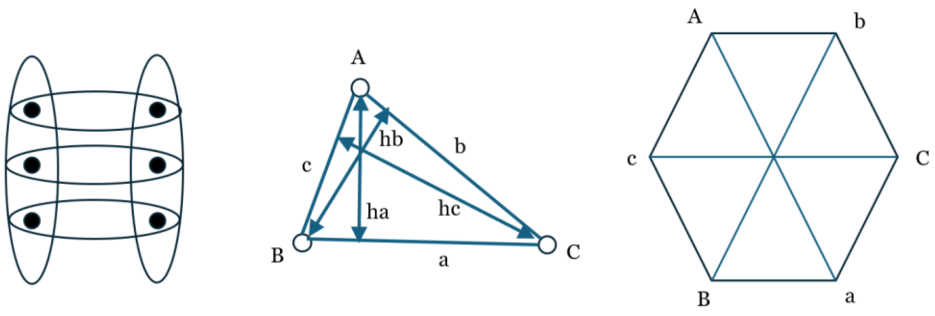

Figure 6: (Left) A minimal well-constrained configuration that consists of five rigid bodies in 2D and contains no triangles. (Middle) A constraint problem in 2D: Derive the geometry of a triangle given its three altitudes and (right) the corresponding constraint graph that contains no triangle (each edge is a rigid body

We propose to systematically generate all such cases of minimal, well-constrained rigid body graphs along with the placement of geometries. Then, one can introduce machine learning approaches for detecting minimal well-constrained rigid body graphs based on graph density with minCutPool layers with spectral clustering in graph neural networks [2] (unsupervised learning) or for detecting certain graph patterns with neighborhood kernels in GNNs [6] (supervised learning).

To increase the scope of our approach, we may (i) include composite geometric objects by transforming the composite object to a set of direct and indirect constraints and elementary geometries and (ii) extend the repertoire of elementary geometries (e.g., circle of variable radii).

4.2 The Root Selection Problem: AI-assisted Root Navigation

A well-constrained geometric constraint system for which we have derived a construction sequence based on graph analysis with m construction steps and an average of k discrete per construction steps will have $O(k^m)$ discrete solutions. Fig. 7 illustrates a geometric constraint problem with three discrete real solutions (up to rotation and translation). Even the more generic problem of computing a real solution for a solvable, well-constrained problem was shown to be NP-hard in a strong combinatorial sense [9].

We propose developing a user-friendly and powerful tool for the root navigation problem employing deep learning. We will use a hybrid GNN-LSTM with attention architecture to choose the appropriate solution based on the construction sequence. There will be a training phase with hundreds of construction sequences from design libraries.

We have experimented with an encoder-decoder LSTM scheme with attention that is capable of capturing a geometric sequence and providing the final placement for all geometric elements. The network is shown in Figure 9. It receives as input the geometric construction sequence (the steps of placing the geometric elements with respect to each other) and the initial geometry placement. Usually, the initial geometry placement represents the geometric elements as determined implicitly by the designer when creating the (incorrect) draft configuration before enforcing the imposed geometric constraints. We have trained the network with 40 different geometric sequences with several constraint value configurations and initial geometry placements.

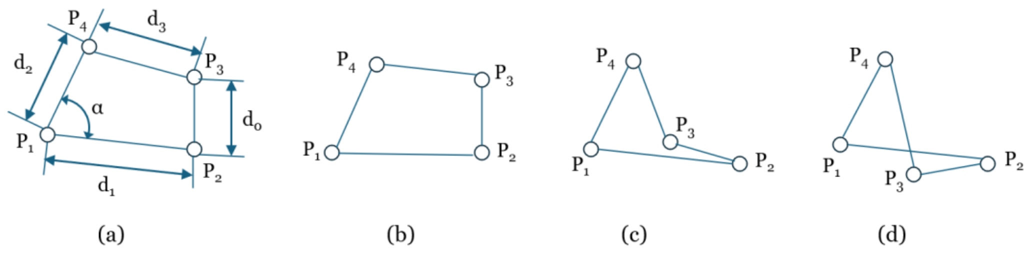

Figure 7: (a) A geometric constraint problem consisting of four points, four straight segments, four point- point distances, and an angle. (b) and (d) Two different solution instances to the constraint problem (a). By changing the value of d_3 one may also get solution (c) which yields a non-intersecting non-convex closed polyline. In most cases (b) captures design intent since it is the only non-intersecting convex closed polyline.

For the construction steps we have used a small set of constructions that corresponds to placing a line or point in 2D from two point/line geometries (5 different construction steps, we cannot place a line in 2D from two other lines since one angle is redundant). We have observed the following:

(i) The network generalizes well to sequences with the same number of construction steps.
(ii) As we increase the training set, the accuracy of the final placement increases. Probably this is because the network learns how to perform each type of placement correctly.
(iii) The network learns to maintain convexity, thus avoiding the case of Figure 7 (c)
(iv) The networks learn to enforce the non-self-intersectingness of closed polylines, thus avoiding the case of Figure 7 (d), which derives a non-manifold object when used as a profile path for extrusion in Figure 8 (right).
(v) Since we provide only symbolic information in the geometric construction sequence, we may use this scheme for a multitude of placement steps, including non-trivial 2D and 3D configurations that require numerical constraint-solving techniques.

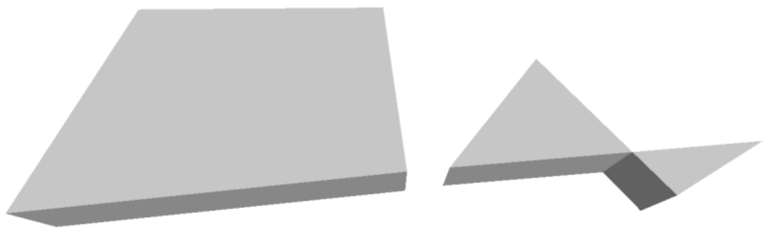

Figure 8: Two solutions for extrusion of the profile of Fig. 7. (left) The solution that corresponds to the profile of Fig. 7(b) and (right) the solution that corresponds to the profile of Fig. 7(c).

5 CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

Geometric Constraint Solving (GCS) undeniably plays a crucial role in Computer-Aided Design (CAD) by ensuring the generation of accurate, precise, and consistent parts. This paper proposes addressing research challenges by integrating traditional GCS methods with graph theory and machine learning approaches. The paper specifically focuses on tackling the well-constrainedness issues in 2D and 3D

GCS problems, grounded in the theoretical foundations of GCS. Additionally, the matter of AI-assisted root selection for graph-based constructive constraint solving is explored.

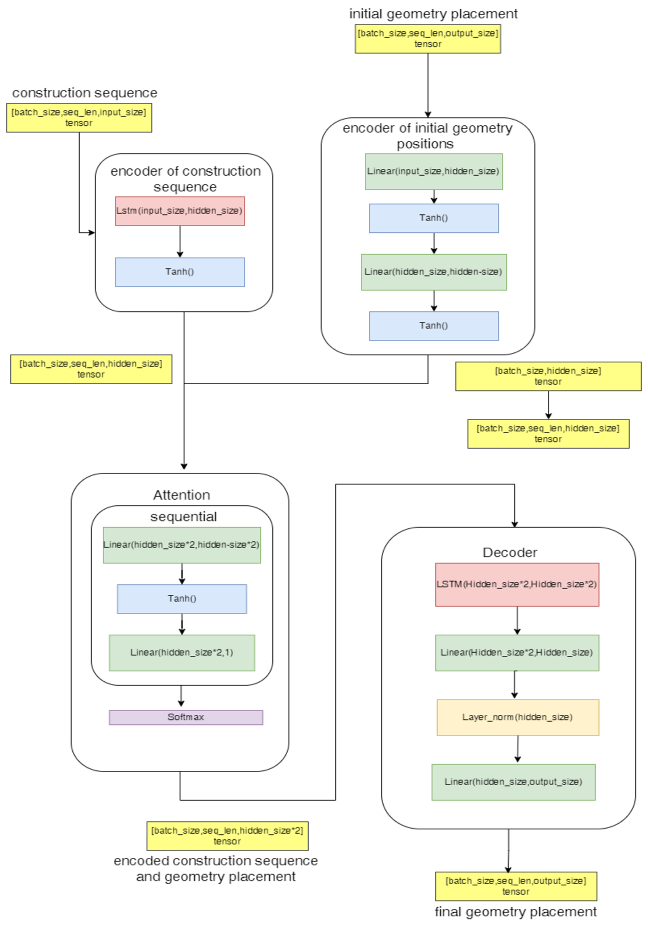
initial geometry placement
[batch_size,seq_len,output_size] tensor
construction sequence
[batch_size,seq_len,input_size] tensor
encoder of initial geometry positions
Linear(input_size,hidden_size)
Tanh()
Linear(hidden_size,hidden-size)
Tanh()
encoder of construction sequence
Lstm(input_size,hidden_size)
Tanh()
[batch_size,seq_len,hidden_size] tensor
[batch_size,hidden_size] tensor
[batch_size,seq_len,hidden_size] tensor
Attention
sequential
Linear(hidden_size*2,hidden-size*2)
Tanh()
Linear(hidden_size*2,1)
Softmax
Decoder
LSTM(Hidden_size*2,Hidden_size*2)
Linear(Hidden_size*2,Hidden_size)
Layer_norm(hidden_size)
Linear(hidden_size,output_size)
[batch_size,seq_len,hidden_size*2] tensor
encoded construction sequence and geometry placement
[batch_size,seq_len,output_size] tensor
final geometry placement

Figure 9: The network architecture for learning to perform geometric construction sequences.

One can employ GCS for placing geometric objects in CAD models. We will investigate cases with (i) simple geometries (points, lines, planes, conics, spheres), (ii) composite parts, and (iii) surface and curve patches.

We also suggest investigating encodings of graph construction sequences for generating novel, meaningful construction sequences (for CAD).

Ioannis Fudos, https://orcid.org/0000-0002-4137-0986
Vasiliki Stamati, https://orcid.org/0000-0002-4225-3685

REFERENCES

[1] Ait-Aoudia, S.; Moussaoui, S.; Abid, K.; Michelucci, D.: On the reducibility of geometric constraint graphs, (2017), arXiv CoRR abs/1712.05578.

[2] Bianchi, F.; Grattarola, D.; Alippi, C.: Spectral Clustering with Graph Neural Networks for Graph Pooling, Proceedings of the 37th international conference on Machine learning, 2729-2738, ACM 2020.

[3] Brehmer, J.; de Haan, P.; Behrends, S.; Cohen, T.: Geometric Algebra Transformer, Advances in Neural Information Processing Systems, 2023, vol. 37, https://arxiv.org/abs/2305.18415.

[4] Brüderlin, B.: Using geometric rewrite rules for solving geometric problems symbolically, In Theoretical Computer Science 116, 1993, pages 291–303. Elsevier Science Publishers B.V.

[5] Durand, C.; Hoffmann, C.M.: A Systematic Framework for Solving Geometric Constraints Analytically, Journal of Symbolic Computation, Volume 30, Issue 5, 2000, Pages 493-519, https://doi.org/10.1006/jsco.2000.0392.

[6] Feng, A.; You, C.; Wang, S.; Tassiulas, L.: KerGNNs: Interpretable Graph Neural Networks with Graph Kernels, Proceedings of the AAAI Conference on Artificial Intelligence 36(6), 2022, 6614–6622, https://doi.org/10.1609/aaai.v36i6.20615.

[7] Foufou, S.; Michelucci, D.: Interrogating witnesses for geometric constraint solving, Information and Computation, 216, 2012, 24–38, https://doi.org/10.1016/j.ic.2011.09.006.

[8] Fudos, I.; Hoffmann, C.M.: Correctness proof of a geometric constraint solver, Intl J Computational Geometry and Applications, 6(4), 1996, 405-420, https://doi.org/10.1142/S0218195996000253.

[9] Fudos, I.; Hoffmann, C.M.: A graph-constructive approach to solving systems of geometric constraints, ACM Trans. On Graphics, 16, 1997, 179-216, https://doi.org/10.1145/248210.248223.

[10] Heydon, A.; Nelson, G.: The Juno-2 Constraint-Based Drawing Editor, Research Report 131a, Digital Systems Research Center, December 1994.

[11] Hoffman, C.M.; Lomonosov, A.; Sitharam, M.: Decomposition Plans for Geometric Constraint Systems, Part I: Performance Measures for CAD, J Symb Comput, 31, 2001, 367–408, https://doi.org/10.1006/jsco.2000.0402.

[12] Fudos, I.; Hoffmann, C.M.; Juan-Arinyo, R.: Tree-decomposable and Underconstrained Geometric Constraint Problems, in: M. Sitharam, A. St. John, J. Sidman (Eds.), Handbook of Geometric Constraint Systems Principles, 1st ed., Chapman and Hall/CRC, Chapter 6, 2017, https://doi.org/10.1201/9781315121116.

[13] Joan-Arinyo, R.; Soto, A.: A correct rule-based geometric constraint solver, Comput Graph, 21(5), 1997, 599–609, https://doi.org/10.1016/S0097-8493(97)00038-1.

[14] Lamure, H.; Michelucci, D.: Solving geometric constraints by homotopy, in C. Hoffmann, J. Rossignac (Eds.), 3rd Symposium on Solid Modeling and Applications, ACM Press, Salt Lake City, Utah USA, 1995, pp. 263–269, https://doi.org/10.1145/218013.218071.

[15] Lee-St.john, A.; Sidman, J.: Combinatorics and the rigidity of CAD systems, Computer-Aided Design, 45(2), 2013, 473–482, https://doi.org/10.1016/j.cad.2012.10.030.

[16] Li, X.; Ge, Q.J.; Gao, F.: A computational geometric approach for motion generation of spatial linkages with sphere and plane constraints, J Mech Robot 11(1), 2019, https://doi.org/10.1115/1.4041788.

[17] Michelucci, D.; Foufou, S.: Geometric constraint solving: The witness configuration method, Computer-Aided Design, 38, 2006, 284–299, https://doi.org/10.1016/j.cad.2006.01.005.

[18] Sridhar, N.; Agrawal, R.; Kinzel, G.L.: Algorithms for the structural diagnosis and decomposition of sparse, underconstrained design systems, Computer-Aided Design, 28, 1996, 237–249, https://doi.org/10.1016/0010-4485(96)88488-0.

[19] Wu, W.: Mechanical Theorem Proving in Geometries, in: B. Buchberger, G. E. Collins (Eds.), Texts and Monographs in Symbolic Computations, Springer Vienna, Vienna, 1994, https://doi.org/10.1007/978-3-7091-6639-0.

[20] Zou, Q.; Feng, H.-Y.: On Limitations of the Witness Configuration Method for Geometric Constraint Solving in CAD Modeling, 2019, arXiv preprint arXiv:1904.00526.