EuclidNet: Deep Visual Reasoning for Constructible Problems in Geometry

Man Fai Wong
City University of Hong Kong
mfwong29-c@my.cityu.edu.hk

Xintong Qi
Columbia University
xq2224@columbia.edu

Chee Wei Tan
Nanyang Technological University
cheewei.tan@ntu.edu.sg

Abstract

In this paper, we present a deep learning-based framework for solving geometric construction problems through visual reasoning, which is useful for automated geometry theorem proving. Constructible problems in geometry often ask for the sequence of straightedge-and-compass constructions to construct a given goal given some initial setup. Our EuclidNet framework leverages the neural network architecture Mask R-CNN to extract the visual features from the initial setup and goal configuration with extra points of intersection, and then generate possible construction steps as intermediary data models that are used as feedback in the training process for further refinement of the construction step sequence. This process is repeated recursively until either a solution is found, in which case we backtrack the path for a step-by-step construction guide, or the problem is identified as unsolvable. Our EuclidNet framework is validated on complex Japanese Sangaku geometry problems, demonstrating its capacity to leverage backtracking for deep visual reasoning of challenging problems.

1 Introduction

Henri Poincaré once remarked "*Geometry is the art of correct reasoning from incorrectly drawn figures*". Thus, reasoning in geometry often means drawing upon visual figures in a creative manner with *abundant trial and error*. This is true especially for geometric constructible problems that analyze the drawing of Euclidean shapes with straightedges and compasses [1, 2]. Every constructible problem requires first a feasibility answer and then the exact sequence of straightedge-and-compass construction steps if feasible. For example, though Gauss proved in 1796 the constructibility of the regular seventeen-sided polygon, its explicit construction remains elusive until decades later as shown by J. Erchinger and H. W. Richmond [2]. Gelernter's geometry machine in 1959 would need a diagram computer to generate figures to validate its automated geometry theorem proving [3].

Visualization of figures is undeniably helpful to humans in developing intuition and grasping mathematical concepts in geometry. This process is not only perceptual but includes a visual validation step by step using trial and error to sieve through the visual information [4]. Kim in [5] proposed visual reasoning in geometry by drawing an analogy to human reasoning in identifying visual patterns and analyzing geometric feature information repeatedly. Other related works include the diagram reasoning in [6], program synthesis technique in [7] to synthesize geometric constructions, the alignment of visual and textual cues in geometrical diagrams [8] and geometric deep learning [9]. It has become increasingly important to understand how to leverage the availability of large data sets of visual figures to refine and accelerate the trial and error process in *automated visual reasoning*.

36th Conference on Neural Information Processing Systems (NeurIPS 2022) Workshop on Math-AI.

In this paper, we propose EuclidNet, a deep learning-driven framework that utilizes neural network-empowered visual reasoning techniques for constructible problems in geometry. In particular, the EuclidNet framework leverages the neural network Mask R-CNN to extract visual features from the images and categorize these visual features into points, lines, and circles. Since intersections may also possess significantly important geometric features, we also implement another Mask R-CNN to extract intersections from the image and count them as points. Together, these features form a space where feasible construction steps are derived and form a rooted tree to traverse all possible features. From a variety of features in the tree, EuclidNet selects one geometrical feature to construct and then adds the construction step to the existing construction. Repeating the process, EuclidNet proposes sequences of constructions where backtracking is conducted once an existing construction is identical to the original input image, and a step-by-step reconstruction geometrical solution can thus be derived from the process. This procedure of divide-and-conquer and backtracking is reminiscent of Wang's algorithm for automated theorem proving of propositional logic [10–12], where a statement is decomposed into multiple premises that are considered true, and the automated program exhausts the solution space to determine if the conclusion is valid.

In summary, the contributions of our work are as follows.

- We offer a new perspective on geometric construction using visual reasoning as a comprehensive procedure to discover solutions. Visual reasoning leverages the feature representation of geometric patterns extracted by Mask R-CNN to propose new construction possibilities.
- We introduce backtracking as a means of step-by-step construction solution finding. Once the existing construction is identical to the original input image, a solution is found, and EuclidNet backtracks the construction sequence to reproduce a *proof* of construction.
- We illustrate EuclidNet's relative performance against other approaches to solving geometric construction problems. We demonstrate the effectiveness of our proposed framework for complex geometric construction problems such as Japanese Sangaku geometry [13].

2 Methodologies

2.1 Elementary Euclidean Constructions Procedure

In a Euclidean plane, points, lines and circles are considered the fundamental constructing elements, which are also known as primitives. When we solve the Euclidean geometric construction problems, it is assumed that specific points are known as priori in the infinite Euclidean plane [14]. With line and circle tools, only limited moves can be made with the existing points in the Euclidean plane:

1. Given two non-identical points $A(x_1, y_1)$ and $B(x_2, y_2)$, we can draw the unique straight line $L = AA$ with the straightedge containing both points. The ray-line will be projected to the canvas boundary by the slope $m = \frac{y_2 - y_1}{x_2 - x_1}$ and y-intercept $b = y_1 - mx_1$ or $b = y_2 - mx_2$.
2. Given two points $C(h, k)$, M and $|CM| > 0$, we can use the compass to draw a unique circle $c = \{C; |CM|\}$ with C being the center, $|CM|$ being the radii, and M on the circle.

For each constructible problem, when new lines or circles are created on a diagram, EuclidNet aims to localize the intersections in each diagram from previous moves to create more points so as to explore other possible new constructions. Once we have exhaustively examined all possible actions in the given construction using the rules above, we are able to build the solution by considering all possible moves with a backtracking algorithm. This procedure is elaborated in Appendix 5.2.

2.2 Deep Visual Reasoning

Although visual reasoning is second nature to humans given the visual inputs [5], it is not trivial for computers to emulate this. Computers need to first sieve through geometric patterns that are visually depicted, and then comprehend logical relationships between patterns (e.g., isometries) in order to reason. Our approach is to employ deep learning-based geometric primitive extraction as well as image segmentation for pattern-seeking so as to enable data-driven pattern recognition for geometrical diagrams. In some sense, the interpretability of the trained machine learning model for automated visual reasoning is to turn geometrical diagrams into *proof without words* for human understanding. In EuclidNet, we implement our deep visual reasoning functionality with Mask R-CNN.

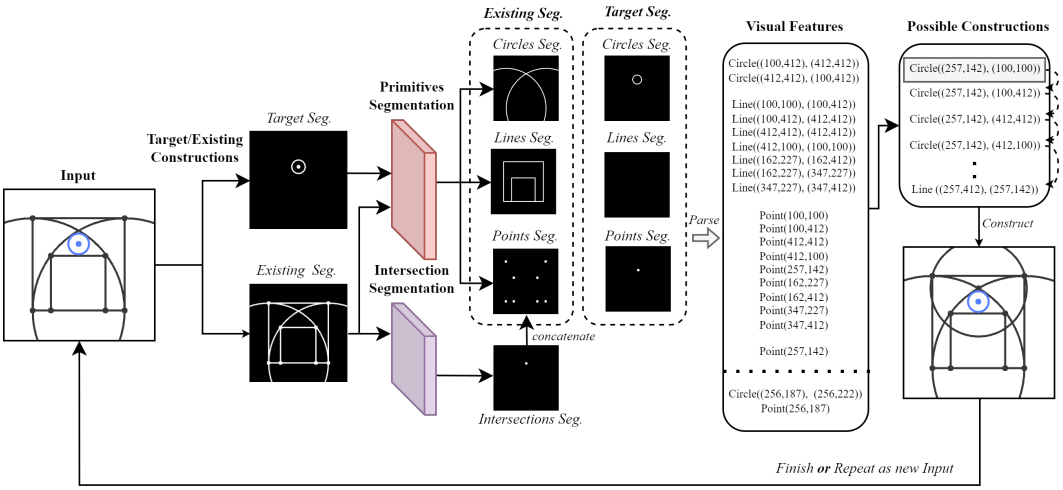


Figure 1: The detailed architecture of the proposed framework (EuclidNet) for solving geometric construction problems leverages visual reasoning and the backtracking algorithm. EuclidNet separates the image as target (in blue) and existing (in black) construction for an input image. EuclidNet consists of two pre-trained segmentation and localization models for geometric primitives and intersections respectively. The segmentation model is based on Mask R-CNN, which processes the input image and performs instance segmentation on the target and existing constructions segments. These pre-trained models receive the images of both the segments and the outputs. The outputs will then be parsed as visual features with their primitive type and location plotted on canvas. All permutations and combinations of the visual features will then be considered as possible construction moves in the next stage. This is achieved through a backtracking algorithm, where EuclidNet tries to build a solution by picking one possible construction move at a time as a new input or backtracking if it reaches the pre-defined maximum search depth. Since the goal construction is extracted during the segmentation process, our framework also examines new possible constructions that contain the targeted geometries. If a solution is found, the framework stops the procedure and saves the current search path. This entire procedure will repeat until there are no new possible constructions.

2.3 Backtracking Algorithm

EuclidNet performs an exhaustive search on the space of all possible moves to enumerate all possible compositions of tools and verifies if any of the constructions match the given target construction. In an exhaustive search, we develop a backtracking algorithm to construct the geometries recursively to build the solution incrementally, one possible move at a time, and we remove those constructions that fail immediately. The final solution is derived from tracing backward and assembling the steps together to form an integrated whole. This backtracking procedure is described in Figure 1, and the detailed implementation of the algorithm can be found in Appendix 6.

2.4 Diagram Element Segmentation and Localization

In our framework, we implement our EuclidNet for the segmentation and localization of primitives and intersections on top of Mask R-CNN with various backbones [15], i.e., Resnet-50 and Resnet-101. First, Mask R-CNN utilizes a regional proposal network to generate regional proposals of the images and determine whether the anchors are foreground or background. Next, the feature map and generated region proposals are forwarded to RoI align to generate a fixed-size feature map. Finally, three tasks are trained simultaneously: classification, box regression, and mask regression, which is performed to increase the accuracy of the mask. The total training loss can be defined as:

$$\mathcal{L} = \mathcal{L}_{cls} + \mathcal{L}_{box} + \mathcal{L}_{mask}, \tag{1}$$

where L_{cls} is the loss of the classification, L_{box} is the bounding-box loss that is identified as those defined in [16], and L_{mask} is the average binary cross-entropy loss including solely the k-th mask if the region is associated with the ground truth class k from [15]. There is a primitives segmentation model for extracting geometric primitives from the target and existing construction

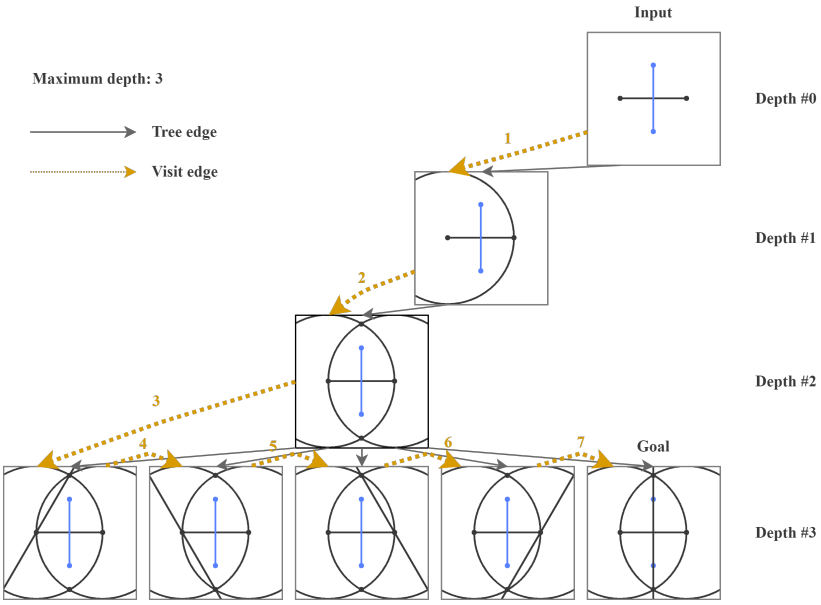


Figure 2: Illustration of the construction of a perpendicular bisector on a line. Given a line with two points as the existing configuration (in black), this problem prompts us to construct a perpendicular bisector (in blue) using only straightedges and compasses. When the search level reaches the maximum depth, the search will backtrack recursively with another tree traversal routine.

images respectively. Since it is still rather difficult to derive insights directly from the result when we construct a new move onto the existing constructions, we also implement an intersection segmentation model to emphasize the intersections of different geometries and treat them as extra points to be concatenated into the point segments, which helps the overall visual reasoning process.

3 Experiment

We build the dataset for primitives and intersections extraction by injecting different primitives and intersections with numbers and scales on images for various experiments with details given in Appendix 7.1. As a performance metric for geometric pattern perception, we adopt the mean average precision (mAP) as our evaluation metric to determine the effectiveness of our segmentation models. Detailed results are presented in Appendix 7.2. We also test EuclidNet with questions in Euclidea [17] to verify its effectiveness in solving geometric construction problems, and we see that it can accurately generate the solutions. Particularly, we also examine how effective EuclidNet is in recognizing patterns like isometries in its construction stages. One instance of the problems solved by EuclidNet is demonstrated in Figure 2. Moreover, EuclidNet demonstrates effectiveness in solving complex geometric construction questions such as the Japanese Sangaku problems [13]. More problems solved by EuclidNet, including two Sangaku problems, can be found in Appendix 8.

4 Conclusion

In this paper, we present EuclidNet, a novel deep learning-driven framework for visual reasoning on geometric constructible problems. The framework localizes geometric primitives using Mask R-CNN models and identifies new intersections from previous moves to discover new constructions. To find the solution, EuclidNet employs a backtracking algorithm to search for possible constructions with relatively low computational complexity. To validate EuclidNet's effectiveness to explore the solution space of constructible problems and its efficacy at reasoning 'constructions', we have conducted various experiments on numerous challenging constructible problems, including Japanese Sangaku geometry. As future work, it is interesting to extend EuclidNet with neural-symbolic artificial intelligence or transfer learning that trains with a minimal amount of data to address geometric construction problems arranged in increasing order of difficulty.

Acknowledgment

The work is supported in part by the Ministry of Education, Singapore, under its Academic Research Fund Tier 1 (Project No. 022307) and Hong Kong ITF Project No. ITS/188/20.

References

[1] David Hilbert. *The Foundations of Geometry*. Open Court Publishing Company, 1902.

[2] Daniel Pedoe. *Geometry, a Comprehensive Course*. Dover Books on Mathematics, 1988.

[3] Herbert L Gelernter and Nathaniel Rochester. Intelligent behavior in problem-solving machines. *IBM Journal of Research and Development*, 2(4):336–345, 1958.

[4] Tommy Dreyfus. On the status of visual reasoning in mathematics and mathematics education. In *Proc. 15th Conf. of the Int. Group for the Psychology of Mathematics Education*, 1991.

[5] Michelle Y. Kim. Visual reasoning in geometry theorem proving. *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, 1989.

[6] Kenneth R Koedinger and John R Anderson. Abstract planning and perceptual chunks: Elements of expertise in geometry. *Cognitive Science*, 14(4):511–550, 1990.

[7] Sumit Gulwani, Vijay Anand Korthikanti, and Ashish Tiwari. Synthesizing geometry constructions. *PLDI on ACM SIGPLAN*, 46(6):50–61, 2011.

[8] Minjoon Seo, Hannaneh Hajishirzi, Ali Farhadi, Oren Etzioni, and Clint Malcolm. Solving geometry problems: Combining text and diagram interpretation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1466–1476, 2015.

[9] M. M. Bronstein, J. Bruna, T. Cohen, and P. Velickovic. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. `https://arxiv.org/abs/2104.13478`, 2021.

[10] Hao Wang. Computer theorem proving and artificial intelligence. In *Automated Theorem Proving: After 25 Years*, volume 29, pages 49—70. Springer, 1984.

[11] Hao Wang. Toward mechanical mathematics. In *IBM Journal*, volume 4, pages 2—22, 1960.

[12] Hao Wang. Proving theorems by pattern recognition. In *Part I, Communications of the Association for Computing Machinery*, volume 3, pages 220—234, 1960.

[13] Hidetoshi Fukagawa and Daniel Pedoe. *Japanese Temple Geometry Problems*. Charles Babbage Research Centre, 1989. ISBN 9780919611214.

[14] Robert Geretschläger. Euclidean constructions and the geometry of origami. *Mathematics Magazine*, 68(5):357–371, 1995.

[15] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2961–2969, 2017.

[16] Ross Girshick. Fast R-CNN. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[17] Euclidea. URL `https://www.euclidea.xyz`.

[18] Pan Lu, Ran Gong, Shibiao Jiang, Liang Qiu, Siyuan Huang, Xiaodan Liang, and Song-Chun Zhu. Inter-GPS: Interpretable geometry problem solving with formal language and symbolic reasoning. In *The 59th Annual Meeting of the Association for Computational Linguistics*, 2021.

[19] Wen-Dun Tun. On the decision problem and the mechanization of theorem-proving in elementary geometry. *Scientia Sinica*, 21(2):159–172, 1978.

[20] Xiao-Shan Gao and Shang-Ching Chou. Solving geometric constraint systems. II. a symbolic approach and decision of RC-constructibility. *Computer-aided design*, 30(2):115–122, 1998.

[21] Jaroslav Macke, Jiri Sedlar, Miroslav Olsak, Josef Urban, and Josef Sivic. Learning to solve geometric construction problems from images. In *International Conference on Intelligent Computer Mathematics*, pages 167–184. Springer, 2021.

[22] Hidetoshi Fukagawa and Tony Rothman. *Sacred Mathematics: Japanese Temple Geometry*. Princeton University Press, 2008. ISBN 9780691127453.

5 Appendix

5.1 Related Works

5.1.1 Computational Geometry

Automated Geometric Reasoning with Formal Reasoning Automated geometric reasoning refers to the process of deriving theorems from geometric problems with computers, which was first attempted by Gelernter and his collaborators in 1959 [3]. Existing automated geometric reasoning methods primarily utilize formal reasoning, which relies on rules of logic and mathematics to conduct reasoning. These methods can be further categorized into synthetic and algebraic approaches. For synthetic reasoning, geometric definitions and theorems are built inside machines, which then incorporate search techniques, often exhaustive, to find the solution [7]. Examples of the synthetic approach can be found in Inter-GPS [18] and GEOS [8], which are solvers with built-in Euclidean formulas. The algebraic approach involves algebraic operations such as Wenjun Wu's method [19] that decomposes problems based on the well-ordering principle and successive pseudo-reduction. However, the algebraic approach has many limitations, and no geometric construction solver has adopted this method thus far.

Automated Geometric Reasoning on Constructions An early attempt to perform geometric reasoning on constructions with images was made by Gao [20] to extract information from the diagram and produce a construction sequence. A more recent image-based geometric construction problem solver is built on top of Mask R-CNN for finding geometric constructions with straightedges and compasses in the Euclidea [21]. The solver uses a Mask R-CNN as a recognizer to generate geometric meanings and arrive at the solutions without using formal reasoning. To be more specific, their approach trains the deep learning models by the images from different tools on Euclidea with variations such as rotation and translation as the dataset. However, this image-based solver only performs effectively on the training data from Euclidea.

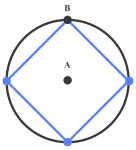
5.2 Geometric Construction with Straightedges and Compasses

Geometric Construction Environment on Euclidea With the burgeoning development of online education platforms, gamified applications like Euclidea [17] are more widely adopted as they have relatively large collections of problems in the game databases. Our geometric construction environment follows a similar setting with only point, line, and circle tools. Some advanced tools available on Euclidea as shortcuts for complex constructions, such as perpendicular bisector, angle bisector, perpendicular, and parallel, can also be constructed using basic tools (see Figure 2). Users are asked to find a sequence of construction steps from an existing configuration (in black) to a given goal configuration (in blue) using the rule and compass.

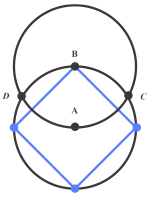
Points of Intersections Since moves in our configuration rely on the existing points in the Euclidean plane, it is required to define new possible points in the plane, i.e., intersections and points from the ray line from the previous moves. For the two non-parallels, there is not necessarily an intersection point for a line segment, but an *line-line intersection* point must exist with their ray. Secondly, we consider the *line-circle intersection*. There are three ways a line and a circle can be associated, i.e., the line cuts the circle at two distinct points, the line is tangent to the circle, or the line misses the circle. Finally, we define the cases on *circle-circle intersection*. If the sum of the radii and the distance between the centers are equal, then the circles touch externally. Also, if the difference between the radii and the distance between the centers are equal, then the circles touch internally. The summary for points of intersections is as follows:

1. Given two non-parallel straight lines l_1 and l_2, we can determine their unique point of intersection $P = l_1 \cap l_2$.
2. Given a circle $c = \{C; r\}$ and a straight line l, the distance between C and l less than or equal to r, there will be either two points or one point of intersection between c and l.
3. Given two circles $c_1 = \{C_1; r_1\}$ and $c_2 = \{C_2; r_2\}$ such that neither contains the center of the other in its interior, and the distance between the centers is less than or equal to the sum of the radii, then there will be two points of intersection between c_1 and c_2.
4. Given two circles $c_1 = \{C_1; r_1\}$ and $c_2 = \{C_2; r_2\}$ such that one contains the center of the other in its interior, and the distance between the centers is no less than the difference between the two radii, there will be at most four points of intersection between c_1 and c_2.

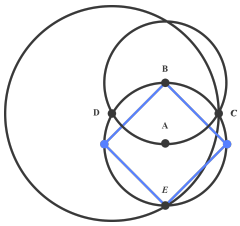
Construction on Euclidea Puzzle: Inscribed Square Figure 3 illustrates a solution to a problem from Euclidea (Alpha-7) with the initial configuration in the black channel and the goal configuration in the blue channel. Users are asked to draw the inscribed square of a circle given the circle with its center. In the original setting of Euclidea, we can solve this problem in 6 steps with shortcut tools such as the perpendicular bisector. However, we show an optimal solution for this problem with only line and circle tools. At each step, the black channel is updated and registered as the current state of the problem. This process will repeat until the current state is identical to the goal configuration.



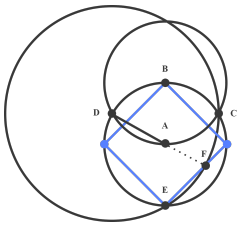
Input: Euclidea *Alpha*-7 (Inscribed Square): Given a circle with center A and a point B on circle, inscribe a square in the circle.



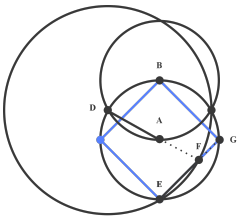
Step 1: Draw a circle from A as a center to B. Two points C and D from circle-circle intersection will be obtained.



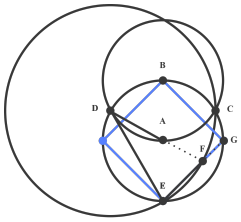
Step 2: Draw a circle from D as a center to C. A point E from circle-circle intersection will be obtained.



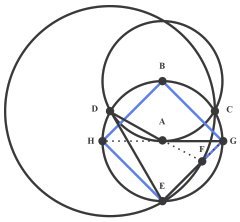
Step 3: Draw a line from D to A, a point F will be obtained from a ray of DA.



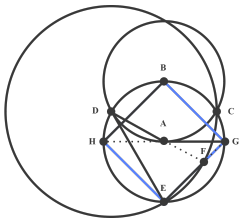
Step 4: Draw a line from E to F, a point G will be obtained from a ray of EF.



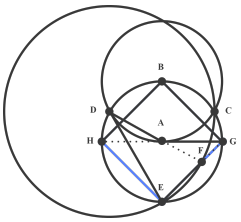
Step 5: Draw a line from D to E.



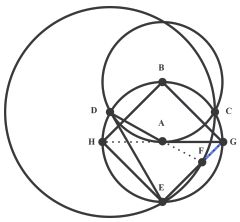
Step 6: Draw a line from G to A, a point H will be obtained from a ray AG.



Step 7: Draw a line from H to B.



Step 8: Draw a line from B to G.



Step 9: Draw a line from H to E and the construction is completed.

Figure 3: Illustration of construction on the inscribed square in a circle using straightedges and compasses. EuclidNet uses the same geometric construction environment described above.

6 Detailed Implementation

6.1 Deep Visual Reasoning with Backtracking

The search algorithm in our framework takes an image of a constructible problem and an empty sequence of images as the initial input. The image of the constructible problem follows the same environment defined in appendix A.2 with the initial configuration in the black channel and the goal configuration in the blue channel. The primitives and intersections information will be extracted by the pre-trained segmentation models for each input image of the search simultaneously. Alternatively, the information can be carried forward to the next depth and skip the extraction during the search. However, since the search is implemented on top of backtracking, it may be expensive to carry forward with all the information when the problem is too complex. The search will stop if a solution is found, and the sequence of images will then be saved. The implementation of the algorithm is defined as follows.

Algorithm 1 (**DVRB**) **D**eep **V**isual **R**easoning with **B**acktracking

Input: Image I_{curr}, Depth D_{curr}, Sequence of Images $\boldsymbol{S}$
Output: Sequence of Images $\boldsymbol{S}$
Variables: Possible Moves $\boldsymbol{M}$, Points $\boldsymbol{P}$, Lines $\boldsymbol{L}$, Circles $\boldsymbol{C}$
Parameters: Goal Image I_{goal}, Maximum Depth D_{max}
Models: Primitives Seg_p , Intersections Seg_i ▷ Pre-trained segmentation models
if $I_{curr} = I_{goal}$ **then** ▷ Check I_{curr} is solved
  save($\boldsymbol{S}$) ▷ Save the solution
  return
else
  if $D_{curr} = D_{max}$ **then**
    return ▷ Reach the maximum depth
  else
    $\boldsymbol{P}, \boldsymbol{L}, \boldsymbol{C} \leftarrow Seg_p$.extract($I_{curr}$) ▷ Extract primitives
    $\boldsymbol{P} \leftarrow \boldsymbol{P}$ + Seg_i.extract(I_{curr}) ▷ Extract intersections for existing points
    $\boldsymbol{M} \leftarrow$ construct($\boldsymbol{P}$) ▷ Construct all moves defined in section 2.1
    $\boldsymbol{M} \leftarrow \boldsymbol{M}$.remove($\boldsymbol{L}, \boldsymbol{C}$) ▷ Remove existing lines and circles from possible moves
    while $\boldsymbol{M}$ not empty **do**
      $m \leftarrow M_0$ ▷ Pick the first possible move in $\boldsymbol{M}$
      $\boldsymbol{S} \leftarrow I_{curr} \oplus m$ ▷ Add the concatenation of a new move and current image
      DVRB($I_{curr} \oplus m, D_{curr} + 1, \boldsymbol{S}$) ▷ Substitute new moves to next depth
      $\boldsymbol{S}$.remove($I_{curr} \oplus m$) ▷ Remove a tested construction
      $\boldsymbol{M}$.remove(m) ▷ Remove a tested move

6.2 Network Architecture

We implement our EuclidNet for the segmentation and localization of primitives and intersections on top of Mask R-CNN [15]. The input RGB image will be divided into two images for each channel and carried forward to the network separately. The detailed architecture is summarized as follows:

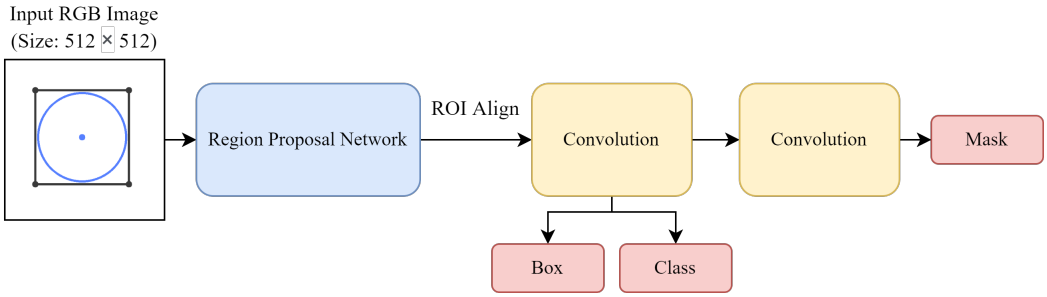


Figure 4: Illustration of Mask R-CNN architecutre in our framework.

7 Experimental Details

7.1 Experiment Setup

We evaluate the performance of our models in terms of the mean average precision (mAP) for both primitives and intersections. For the backbone network, we experiment with ResNet-50 and ResNet-101. To train segmentation models, we set the SGD optimizer with a learning rate of 0.001 and the minimum detection confidence as 0.7. The training is launched on a single NVIDIA 2080Ti GPU (11GB) with a batch size of 16. Other parameters follow the default configuration in [15]. The models are trained on 1000 simulating images in Figure 5 with primitives and intersections separately on 200 epochs and 1000 steps per epoch. We also evaluate the accuracy of EuclidNet on the first 6 levels of Euclidea.

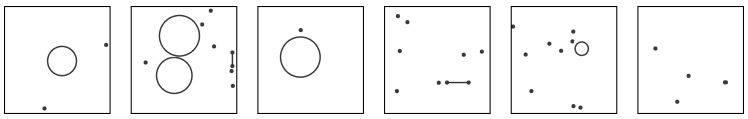


Figure 5: Illustration of injecting different primitives and intersections with numbers and scales.

7.2 Experiment Results

Table 1 shows the performance of the trained Mask R-CNN with different backbones. The result shows that our models can detect the targets with mAP over 90%. The illustration of the detection results for Euclidea problems is shown in Figure 6. Euclidea does not yet support questions involving the area, but it works well on the other constructible problems. We show the accuracy of solving geometric construction problems on the first six levels of Euclidea in Table 2.

Table 1: PERFORMANCE COMPARISON: MASK R-CNN WITH BACKBONES RESNET-50 AND RESNET-101 ON THE PRIMITIVES AND INTERSECTION SEGMENTATION.

Backbone	**Primitives**				**Intersections**			
	mAP	AP_{50}	AP_{75}	AP_{90}	mAP	AP_{50}	AP_{75}	AP_{90}
ResNet-50	0.936	0.949	0.945	0.868	0.926	0.911	0.908	0.901
ResNet-101	0.938	0.951	0.948	0.866	0.928	0.914	0.912	0.906

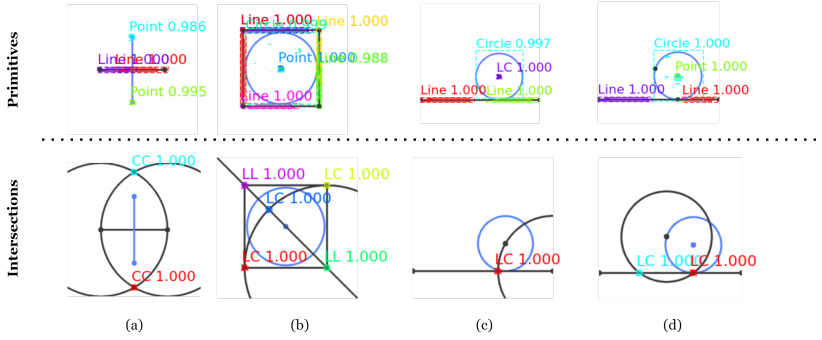


Figure 6: Illustration of segmentation for primitives and intersections. (a) Perpendicular Bisector (b) Circle in Square (c) Circle Tangent to Line (d) Circle through a Point Tangent to Line

Table 2: EVALUATION RESULT ON THE EUCLIDEA DATA SET

	Alpha	**Beta**	**Gamma**	**Delta**	**Epsilon**	**Zeta**
Accuracy	0.857	0.900	0.778	0.636	0.727	0.636

8 Computational Examples

8.1 Euclidea Puzzle (*Alpha-4*): Inscribed Circle

Given a triangle, an inscribed circle is the largest circle contained within the triangle. The inscribed circle will touch each of the three sides of the triangle at exactly one point.

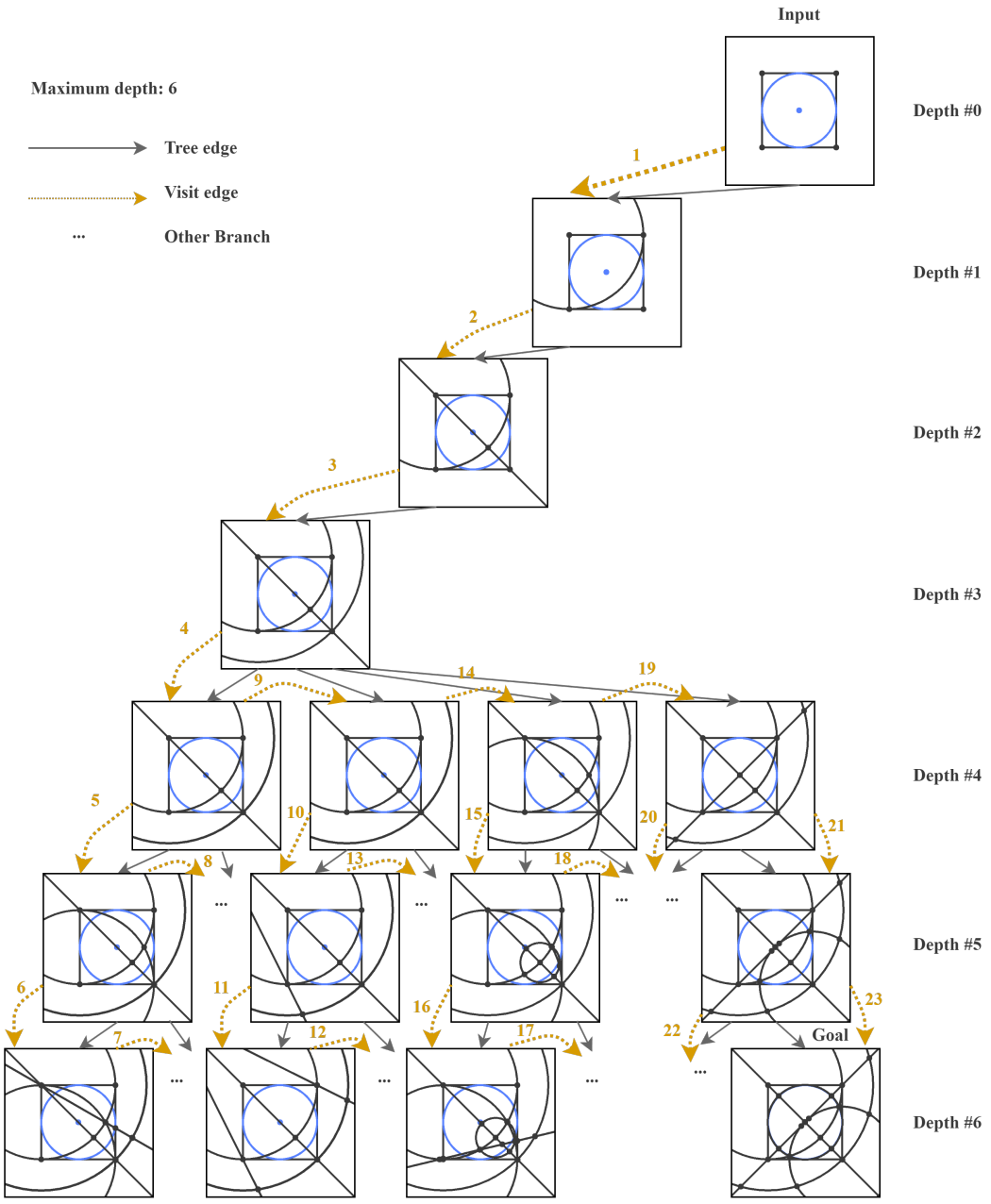


Figure 7: Illustration of the construction of a circle inscribed in the square (Euclidea *Alpha-4*). The maximum depth in this example is 6.

8.2 Euclidea Puzzle (*Gamma-5*): Circle Through a Point Tangent to Line

A circle through a point tangent to the line is to construct a circle through the arbitrary point that is tangent to the given line at the point on the line.

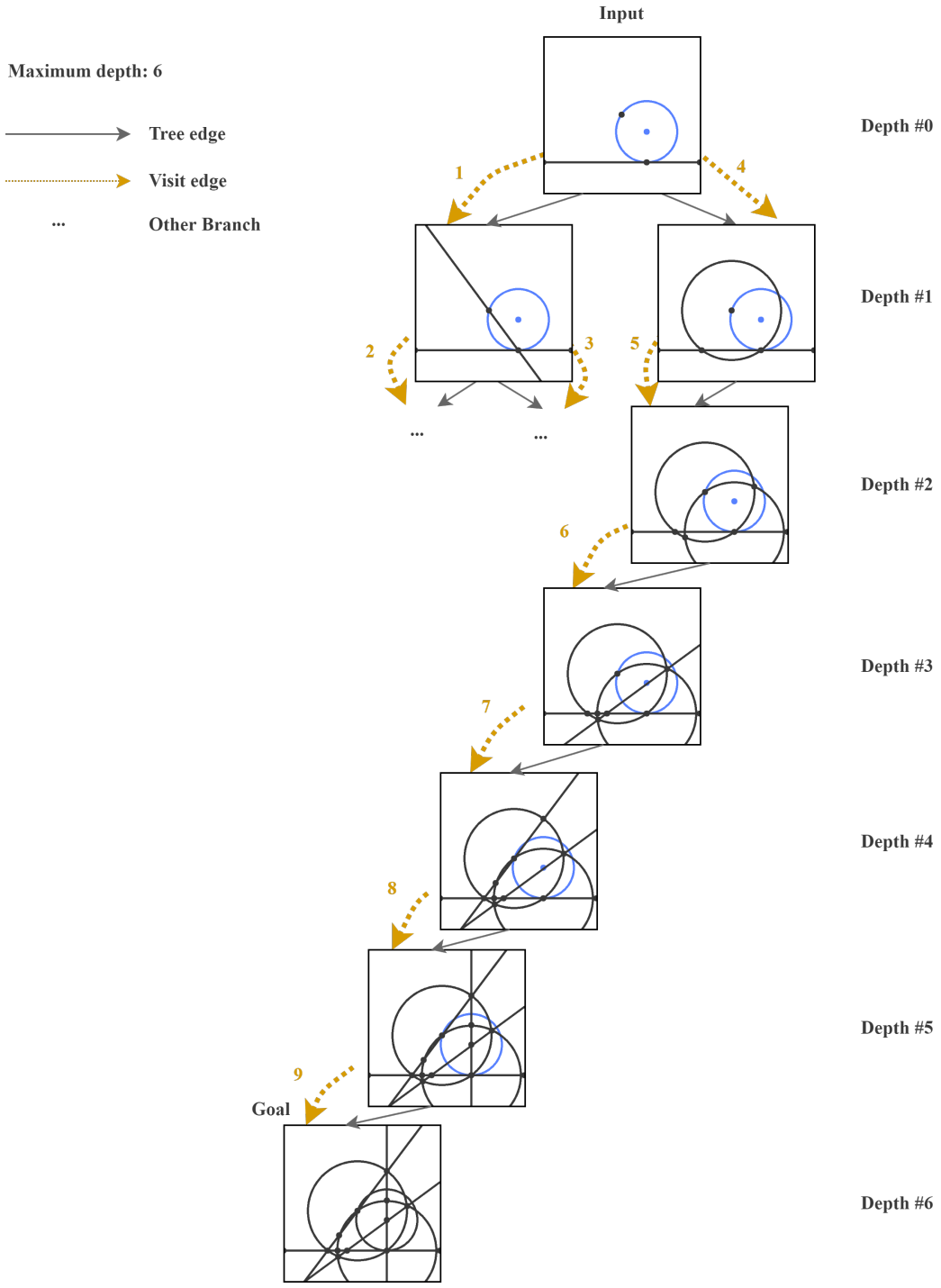


Figure 8: Illustration of the construction of a circle through a point tangent to line. The maximum depth in this example is 6.

8.3 Sangaku: Square and Circle in a Gothic Cupola

Square and Circle in a Gothic Cupola from [22] are a Sangaku with two-quarter circles inscribed in a square form a gothic cupola. Inscribed in the latter is a circle on top, which stands a small circle.

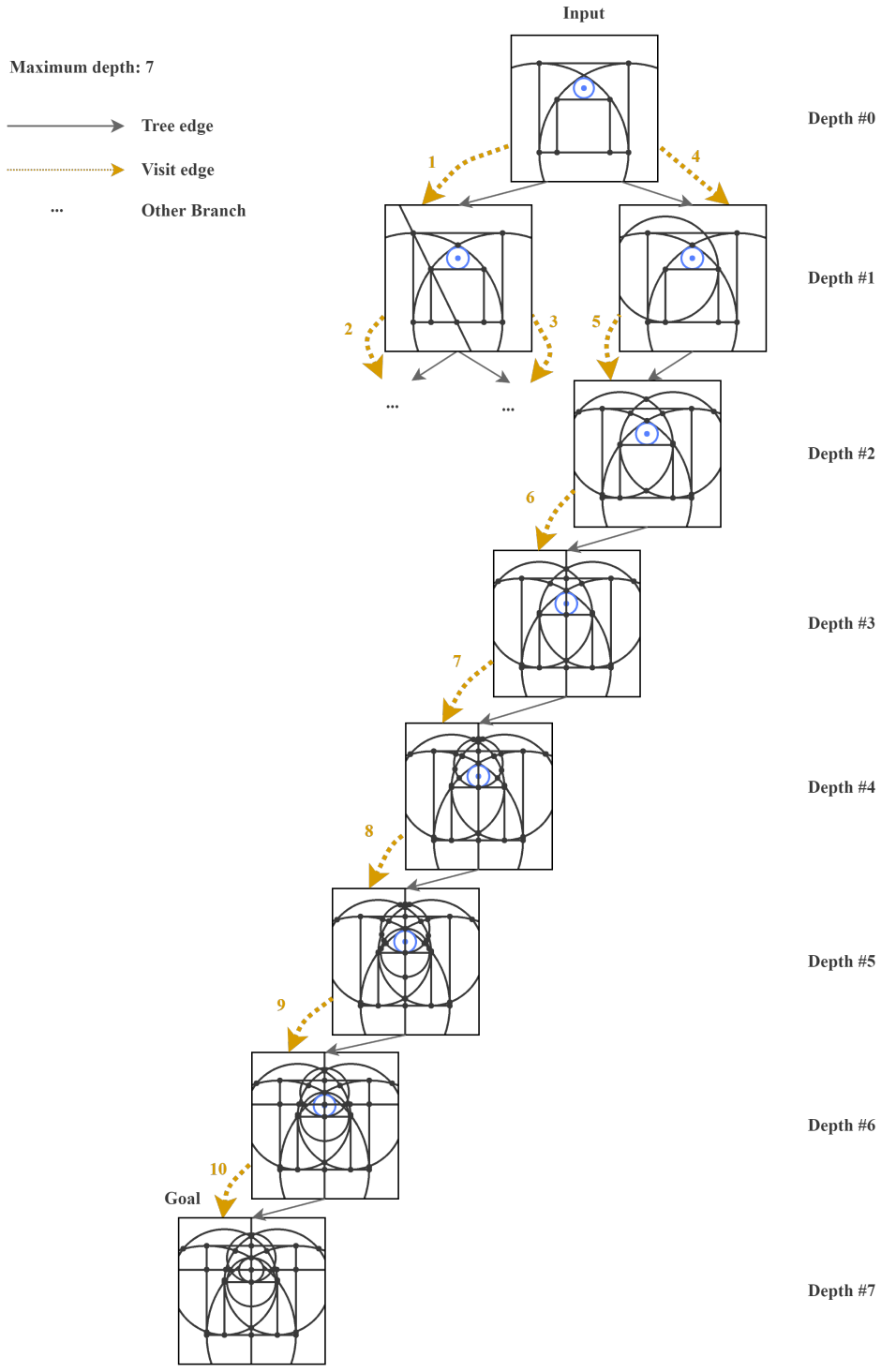


Figure 9: Illustration of the construction of a square and circle in a Gothic Cupola. The maximum depth in this example is 7.

8.4 Sangaku: Sangaku with Versines

Sangaku with Versines was written in 1825 in the Tokyo prefecture from [22]. It relates to the rarely used nowadays versine quantities and the distance from a vertex to the incircle.

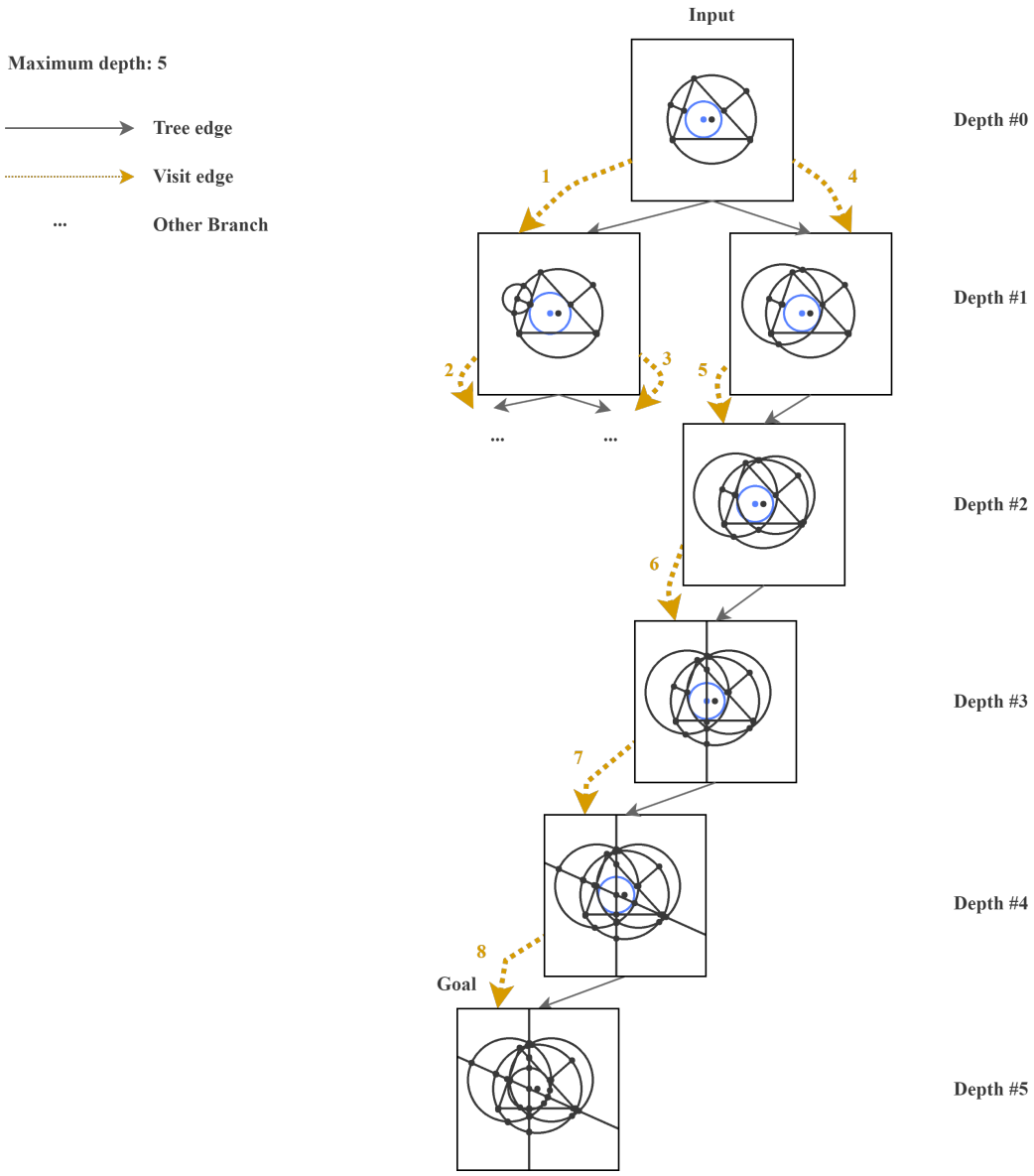


Figure 10: Illustration of the construction of Sangaku with versines. The maximum depth in this example is 5.