JOURNAL OF
EMERGING INVESTIGATORS

Article

An efficient approach to automated geometry diagram parsing

Nikhil Date[1], Sachin Date[2]

[1] Lilavatibai Podar High School, Mumbai, Maharashtra, India

[2] e-Emphasys Technologies, Mumbai, Maharashtra, India

SUMMARY

An automated geometry problem solver can be a valuable tool in math education as a learning aid. An essential part of such a system is the ability to parse diagrams automatically, and diagram understanding by itself is an interesting research problem because of the rich and complex information that geometry diagrams convey and the many approaches one can take to extract that information. In this paper, we introduce Fast Geometry Diagram Parser (FastGDP), an efficient approach to geometry diagram understanding that uses clustering and corner information. We hypothesized that FastGDP would be significantly faster than the widely used GeoSolver tool at both primitive (line and circle) and point detection, because FastGDP does not require large numbers of computationally expensive pixel-level calculations. We further hypothesized that FastGDP would offer comparable performance to GeoSolver on point detection, due to FastGDP's use of corner information. We expected FastGDP's primitive detection performance to be marginally lower than that of GeoSolver due to the latter's emphasis on over-generation of primitives and subsequent selection of the best detections. Our experiments on three datasets (combined n=169) showed that FastGDP is more than an order of magnitude faster than GeoSolver in most cases. We found that FastGDP reports comparable performance to GeoSolver on primitive detection and slightly lower performance on point detection. We believe that the speed advantage offered by FastGDP will provide greater flexibility when it is incorporated into an automated geometry problem solver, especially if FastGDP is used within the training loop of the solver.

INTRODUCTION

An automated geometry problem solver, which is a system that can automatically determine the solution to a geometry problem given the problem text and associated diagram, holds potential for application in artificial-intelligence assisted math education, especially if it can produce human-interpretable solutions. Such a system must have two essential capabilities: the ability to understand the problem text and the ability to parse geometry diagrams. These capabilities are usually achieved using artificial intelligence or machine learning techniques. Geometry diagram parsing in particular is an essential part of a geometry-problem solver as geometry diagrams often provide information that is missing in the problem text. Furthermore, diagrams provide a way to test the validity of inferences drawn from the problem text.

A geometry diagram parsing approach was proposed by Chen, *et al.* in 2014 (1). This approach uses the Hough Transform for both line and circle detection with some additional heuristics (2, 3).

The first widely used geometry diagram parser, which they called G-ALIGNER, was announced in 2014, by Seo, *et al.* (4, 5, 6, 7). Seo, *et al.* then used their diagram parser to design a geometry problem solver called GeoSolver in 2015 (4). Most other geometry problem solvers developed since, such as the tools developed by Sachan, *et al.* and Lu, *et al.*, use the GeoSolver diagram parser (5, 6). One of the only publicly available problem solvers that does not use the GeoSolver diagram parser is the Neural Geometry Solver (NGS), developed by Chen, *et al.* (8). However, NGS does not explicitly detect lines, circles, or points but rather uses a pretrained neural network to extract a feature matrix from the diagram image. In addition to Seo, *et al.*'s diagram parser, another method was announced by Song, *et al.* (9) in 2017 specially geared toward hand-drawn diagrams. This approach does not use the Hough Transform and instead skeletonizes the diagram and then works at the pixel level to detect points, lines, and circles. While this approach is effective for hand-drawn diagrams, it is slow compared to Hough Transform-based methods.

The results of Seo, *et al.* and Lu, *et al.* indicate that a large proportion of the error in geometry problem solving algorithms is due to errors in diagram parsing (4, 6). Thus, further research is necessary to improve geometry diagram parsing. As a first step in this direction, we created the Fast Geometry Diagram Parser (FastGDP), a novel and efficient approach to diagram parsing. Unlike the GeoSolver diagram parser, which uses a submodular optimization approach that requires large numbers of computationally expensive pixel-level calculations for primitive (line and circle) detection, FastGDP uses a clustering-based approach to filter out false positives and improve detection accuracy at the same time. Furthermore, unlike GeoSolver, FastGDP uses results from running the Harris corner detector on diagrams to improve point detection performance—for instance, to filter out false positive point detections. In addition, unlike Chen, *et al.*'s

method, FastGDP uses a parameter selection procedure for circle detection, and also uses clustering to remove false positive primitive detections (1).

Like FastGDP, the GeoSolver tool uses the Hough Transform at its heart to detect lines and circles, or primitives. GeoSolver sets the parameters of the Hough Transform so that it over-generates primitives. GeoSolver then uses a submodular optimization approach to select the best primitives based on an objective function, which rewards covering most of the black pixels in the image with primitives and detecting larger lines and circles. This approach requires large numbers of expensive pixel-level calculations to select primitives. In contrast, FastGDP does not over-generate primitives. Instead, FastGDP uses a parameter selection procedure for circle detection and uses Density-Based Spatial Clustering of Applications with Noise (DBSCAN) clustering for both line and circle detection to filter out false positives (10).

In this study, we compared the detection performance and the detection speed achieved by FastGDP for both primitive and point detection to that of GeoSolver. We made the following two hypotheses about the results of our experiments.

As FastGDP does not need to perform large numbers of pixel-level calculations for primitive detection, and the time for point detection is low compared to the time for primitive detection, we hypothesized that FastGDP would be significantly faster than GeoSolver at both primitive and point detection. We expected that FastGDP would perform somewhat worse at primitive detection than GeoSolver because we expected GeoSolver's submodular optimization approach to be more robust, especially in the case of more complex diagrams. This is because it avoids the parameter sensitivity of Hough Transform-based methods by using parameters that always over-generate primitives. On the other hand, as FastGDP uses corner information to filter out false positive point detections and to detect points that would otherwise not be detected due to mistakes in primitive detection, we expected that FastGDP would offer point detection performance comparable to that of GeoSolver.

Our results showed that FastGDP was significantly faster than GeoSolver while achieving comparable performance with GeoSolver (as quantified by the F1 score) on primitive detection, but it achieved slightly lower performance on point detection.

RESULTS

To test the functionality of FastGDP, we compared the performance of FastGDP and GeoSolver on three geometry diagram datasets of varying diagrammatic complexity and containing a wide variety of types of diagrams, which we call Dataset 1, Dataset 2, and Dataset 3 **(Figure 1)**. The first two datasets were used while building FastGDP, while the third consisted of unseen images.

Precision, Recall and F1 Score

In keeping with the metrics used by Seo, *et al.*, we used precision, recall, and F1 scores to quantify the detection performance of FastGDP and GeoSolver (7). To calculate these metrics for a particular diagram and diagram parser, we ran the diagram parser on that diagram and programmatically determined the number of correctly detected primitives and points. We chose to present micro-averaged metrics, as the number of primitives and points tends to vary considerably among different diagrams, and the micro-averaged metric weights each detection equally rather than weighting each diagram equally.

We calculated precision, recall, and F1 from the information about the correctly detected and ground truth points and primitives. The F1 score can be considered to be a combined metric for precision and recall, so the value of the F1 score alone is a good indicator of the detection performance of a particular diagram parser on a particular task.

We found that FastGDP's precision on primitives on Dataset 1 was significantly higher than that of GeoSolver (p-value = 0.0067) **(Table 1)**, while GeoSolver's recall on primitives on Dataset 2 and its recall and F1 score on points on Dataset 3 set were all significantly higher than that of FastGDP. **(Table 2, Table 3)**. In all other cases, we found no statistically significant difference between FastGDP's and GeoSolver's performance.

Between precision and recall, among all three datasets and for both primitives and points, the only metric where FastGDP performed consistently worse than GeoSolver was recall for both primitive and point detection, although this reduction in performance was statistically significant only for point detection on Dataset 3 and primitive detection on Dataset 2 **(Table 2, Table 3)**. Additionally, the lower recall was often compensated for by higher precision, leading to F1 scores that were comparable to those achieved by GeoSolver.

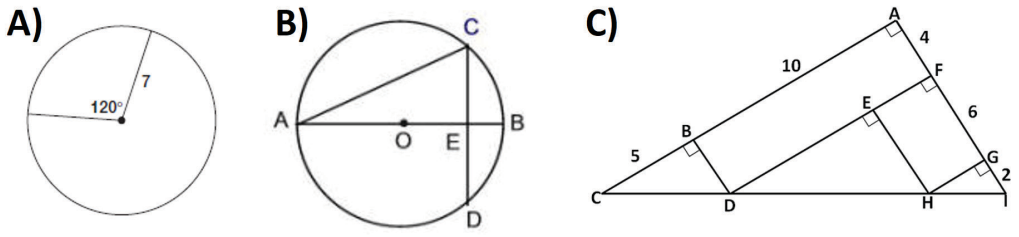

Figure 1: Selected diagrams from the three datasets used in this paper in increasing order of complexity. A) Representative diagram from Dataset 1. B) Representative diagram from Dataset 2. C) Representative diagram from Dataset 3. Diagram A is the least complex and Diagram C is the most complex.

Performance Metric	FastGDP Primitive Detection	GeoSolver Primitive Detection	*p*-value (paired 1-sided)	FastGDP Point Detection	GeoSolver Point Detection	*p*-value (paired 1-sided)
Precision	**97.30%**	87.41%	0.994	**90.88%**	83.89%	.947
Recall	95.45%	**97.35%**	0.094	90.64%	**93.32%**	.062
F1 score	**96.37%**	92.11%	0.890	**90.76%**	88.35%	.451

Table 1: Comparison of FastGDP and GeoSolver detection performance on Dataset 1 (n = 65). Precision, recall, and F1 scores are micro-averaged. The better score for each combination of dataset and metric is bolded. *p*-values less than 0.05 (if any) are also bolded.

Performance Metric	FastGDP Primitive Detection	GeoSolver Primitive Detection	*p*-value (paired 1-sided)	FastGDP Point Detection	GeoSolver Point Detection	*p*-value (paired 1-sided)
Precision	96.90%	**97.98%**	0.351	**90.44%**	88.09%	.779
Recall	89.21%	**92.38%**	**0.029**	88.83%	**90.10%**	.411
F1 score	92.89%	**95.10%**	0.139	**89.63%**	89.08%	.742

Table 2: Comparison of FastGDP and GeoSolver detection performance on Dataset 2 (n = 40). Precision, recall, and F1 scores are micro-averaged. The better score for each combination of dataset and metric is bolded. *p*-values less than 0.05 (if any) are also bolded.

Performance Metric	FastGDP Primitive Detection	GeoSolver Primitive Detection	*p*-value (paired 1-sided)	FastGDP Point Detection	GeoSolver Point Detection	*p*-value (paired 1-sided)
Precision	**99.13%**	98.71%	0.712	**96.85%**	95.81%	.732
Recall	95.42%	**95.83%**	0.295	89.07%	**95.50%**	**.014**
F1 score	97.24%	**97.25%**	0.541	92.80%	**95.65%**	**.023**

Table 3: Comparison of FastGDP and GeoSolver detection performance on Dataset 3 (n = 64). Precision, recall, and F1 scores are micro-averaged. The better score for each combination of dataset and metric is bolded. *p*-values less than 0.05 (if any) are also bolded.

Distribution of F1 Scores

Even though aggregate metrics like micro-averaged precision, recall, and F1 score are useful for assessing the overall performance of a diagram parser on a particular task and dataset, they do not tell the entire story. For instance, two diagram parsers might have the same micro-averaged F1 score on a particular dataset, but the first might achieve perfect results on many more diagrams than the second. Depending on how tolerant to errors in diagram parsing a particular geometry-problem solver is, that geometry problem-solver might perform worse overall if the second diagram parser is used than if the first one is.

This made it important for us to look at the distribution of F1 scores achieved by both GeoSolver and FastGDP on primitive and point detection. For both primitive and point detection, both FastGDP and GeoSolver had similar distributions **(Figure 2)**.

Speed

We also computed the detection time per diagram averaged over five runs (to get a more accurate result) for both FastGDP and GeoSolver on the point detection and primitive detection tasks for each dataset.

On Datasets 1 and 2, FastGDP was more than an order of magnitude faster for both primitive and point detection **(Figure 3)**. On Dataset 3, FastGDP was about 7.3 times faster for primitive detection and around 5 times faster for point detection **(Figure 3)**. All pairwise differences in time were statistically significant **(Figure 3)**. FastGDP's time advantage across all datasets was also statistically significant for both primitive point detection.

DISCUSSION

Our experiments confirmed our hypothesis that FastGDP is significantly faster than GeoSolver at both primitive and point detection. In fact, we found that the overall speed of FastGDP as compared to GeoSolver across all datasets is more than an order of magnitude higher for primitive detection and almost an order of magnitude higher for point detection. An interesting observation was that the time

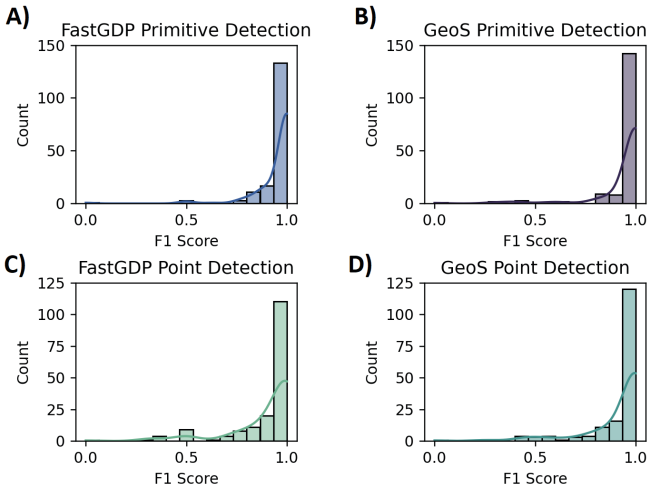

Figure 2: Distributions of F1 scores achieved by FastGDP and GeoSolver for primitive and point detection on all datasets taken together (n = 169). This figure shows histograms and associated KDE plots of the F1 scores achieved by (A) FastGDP on primitive detection, (B) GeoSolver on primitive detection, (C) FastGDP on point detection, and (D) GeoSolver on point detection.

advantage offered by FastGDP seemed to increase with an increase in the complexity of the diagrams. FastGDP had the greatest time advantage on Dataset 2 and the least time advantage on Dataset 3 (which contained the simplest set of diagrams among the three data sets). When diagrams are complex, GeoSolver needs to perform more computationally expensive calculations to detect primitives, which slows it down. Furthermore, we observed that FastGDP provided no additional time advantage for point detection over the advantage already provided for primitive detection. This is consistent with our hypothesis, since GeoSolver performs the most pixel-level calculations for primitive detection.

We also hypothesized that FastGDP will perform somewhat worse at primitive detection but comparably at point detection when compared to GeoSolver. However, the results were contrary to this hypothesis. We found that FastGDP offered comparable performance to GeoSolver on primitive detection, which did not agree with our prediction. However, FastGDP's F1 score for point detection on the Dataset 3 was worse than that of GeoSolver, even though the combined performance on all datasets was not significantly worse. This suggests that the corner detection approach is not as robust as expected on unseen images, and some further work is necessary here.

If overfitting had taken place when FastGDP was being designed, we would have expected it to perform worse on Dataset 3 (containing unseen diagrams) than on the other two datasets. However, as FastGDP actually performed slightly better on Dataset 3 than on the other datasets for both primitive and point detection (perhaps due to the lower complexity of diagrams), we concluded that significant overfitting did not take place when FastGDP was being designed. However, GeoSolver did have a higher F1 score than FastGDP on point detection on Dataset 3 as it outperformed FastGDP in recall.

Another interesting observation was that, while FastGDP's point detection F1 score was lower for Dataset 2 (containing complex diagrams) than for the other two datasets, it was lower to a smaller extent than the primitive detection F1 score. This may suggest that the corner information was helping to

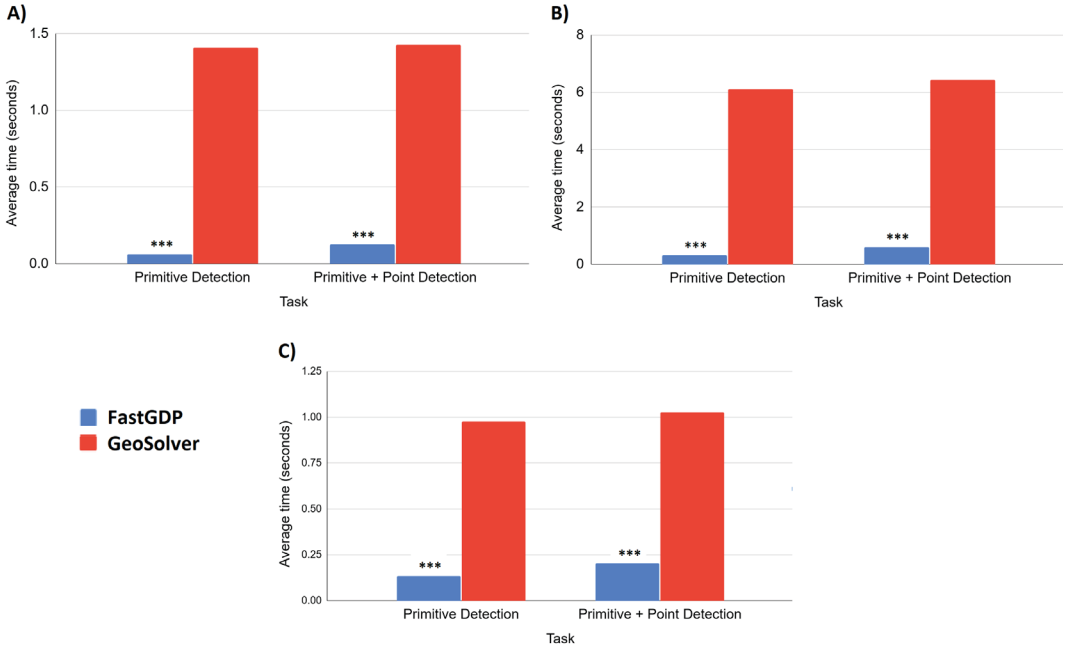

Figure 3: Average time per diagram for Primitive Detection and Primitive + Point Detection for every combination of diagram parser, dataset, and task. This figure shows comparisons of the average time per diagram for both Primitive Detection and Primitive + Point Detection for FastGDP (blue) and GeoSolver (red) on (A) Dataset 1 (B) Dataset 2 and (C) Dataset 3. Each diagram parser was run on each diagram image 5 times for each task and the results of the 5 runs were averaged. $p < .001$ is shown as ***.

compensate for lower primitive detection accuracy.

A limitation of FastGDP is that a large number of spurious corner detections are produced along jagged edges if the image is of very low quality. This can happen especially when the lines and circles are heavily aliased or have very rough edges. It is possible that this could be fixed using preprocessing steps like blurring, but this would have to be performed without losing diagram detail. Additionally, an overly smoothed image might prevent actual corners in the diagram from being detected correctly. Another limitation is that, unlike Song, *et al.*'s method, FastGDP is not designed for parsing hand-drawn diagrams (9).

A limitation of our experiments is that two of the datasets on which we reported data were used when building FastGDP. However, this did not affect the speed results, as FastGDP was not optimized for speed in any way that depended on a particular dataset. In addition, we believed that reporting detection performance data from all three datasets was necessary since the three datasets we used differed in both source and complexity of diagrams. This means that the results on the third dataset alone were not representative of the results on all datasets.

While FastGDP has shown promising results in our experiments, further work is necessary to make use of the advantages of FastGDP in an automated geometry problem solver. The corner detection is not as robust as initially expected, so more research is required to improve the corner detection. If the corner detection accuracy can be significantly improved, it might be possible for FastGDP to consistently outperform GeoSolver in point detection accuracy while still maintaining a speed advantage. A possible way to do this would be to use SMBO (sequential model-based optimization) techniques such as Bayesian Optimization or Tree of Parzen Estimators (TPE) to optimize the parameters of the corner detector (11).

We believe that the speed advantage offered by FastGDP would be particularly beneficial if FastGDP were to be used in the training loop of a larger geometry problem solving system. A significant speed advantage would offer greater flexibility in training a larger model and would also allow the parameters of FastGDP to be tuned simultaneously while training the larger model. None of the currently available geometry problem solvers that use GeoSolver to parse diagrams have attempted to tune GeoSolver's parameters to improve detection accuracy for their specific use cases. Furthermore, a faster diagram parser will make it more feasible to run input images through the diagram parser in every training step instead of precomputing diagram features, which will enable image augmentation to be used in the training process.

In summary, the contributions of this paper to the field of geometry problem solving are two-fold: Firstly, we provide a diagram parser that is significantly faster than the currently available GeoSolver tool while offering comparable performance in most cases. Secondly, we present a novel approach for geometry diagram parsing that adds to the literature on the subject and helps inspire future research.

MATERIALS AND METHODS

Datasets

Dataset 1 contained the training data used by GeoSolver and consisted of 65 geometry diagrams. This dataset was compiled by Seo, *et al.* and consisted of high-school level geometry questions with corresponding geometry diagrams sourced from test-preparation websites such as RegentsPrepCenter, EdHelper, etc. (7). Dataset 2 was a new dataset containing 40 geometry diagrams that we created manually. It consisted of significantly more complex geometry diagrams than those in the first dataset (this dataset did not have associated geometry questions since FastGDP only deals with diagrams). Dataset 3 is the GeoSolver test data containing 64 official SAT® Geometry Questions. The images in this dataset had never been used while designing FastGDP. Dataset 3 contained the simplest geometry diagrams on average, and Dataset 2 contained the most complex ones **(Figure 1)**.

For each dataset, we manually annotated the points and primitives using the Computer Vision Annotation Tool (CVAT) to provide the ground truth line, circle, and point data to compare the results of the algorithms with during testing (12).

Metrics and Evaluation

To test the accuracy of both FastGDP and GeoSolver, we ran each diagram parser on each diagram in the three datasets and then ran code that compared the detections for both primitives and points with the ground truth annotations to determine precision, recall, and F1 score. In doing this, we needed to determine which of the point and primitive detections produced by FastGDP or GeoSolver were correct (i.e, they matched up with a ground truth point or primitive). We did this as follows:

FastGDP uses the Hesse normal form to represent lines, which represents a line by two variables, rho and theta. For line detection, we considered a predicted line to match a ground truth line if the difference between the theta values of the two lines was less than 0.1 radians (5.73 degrees) and the difference between the in rho values was less than 5% of the average dimension of the image. For circle detection, we considered a predicted circle to match a ground truth circle if the IOU (Intersection over Union) was more than 0.8. For point detection, we considered a predicted point to match a ground truth point if the distance between the points was less than 5% of the average dimension of the image. We chose these thresholds for point and line detection as geometry diagrams typically do not have two points or lines respectively that differ by less than the chosen thresholds, so that it would be unlikely for a wrongly detected point or line to be considered correctly detected. The threshold for circle detection was the same as the one used by Seo, *et al.* (7).

We then calculated precision and recall for each diagram parser for each task as follows:

$$precision\ for\ primitives = \frac{Number\ of\ correctly\ detected\ primitives}{Total\ number\ of\ detected\ primitives} \quad (1)$$

$$recall\ for\ primitives = \frac{Number\ of\ correctly\ detected\ primitives}{Total\ number\ of\ ground\ truth\ primitives} \quad (2)$$

$$precision\ for\ points = \frac{Number\ of\ correctly\ detected\ points}{Total\ number\ of\ detected\ points} \quad (3)$$

$$recall\ for\ points = \frac{Number\ of\ correctly\ detected\ points}{Total\ number\ of\ ground\ truth\ points} \quad (4)$$

The F1 score for primitives was calculated using precision and recall for primitives, and the F1 score for points was calculated using precision and recall for points. In each case, we computed the F1 score as follows:

$$F_1\ score\ = 2 \cdot \frac{precision \cdot recall}{precision + recall} \quad (5)$$

To generate the results for speed for some combination of diagram parser, task (primitive or point detection), and diagram, we ran the diagram parser on each diagram five times for that task, measured the time for each run (programmatically), and then averaged the five values to determine the average detection time. We repeated this process for every combination of diagram parser, task, and diagram.

We ran all experiments on a desktop computer having an Intel i7-7700k Central Processing Unit (CPU), integrated graphics, and 32 GB of RAM.

Statistical Methods

When calculating a *p*-value for comparing the performance of FastGDP and GeoSolver on some combination of metric and dataset, we paired up the values achieved by FastGDP and GeoSolver on that metric. We then ran the one-sided paired *t*-test on the paired data using Google Sheets. For precision, recall, and F1, the alternate hypothesis was that GeoSolver's performance was better (i.e. higher), while for average detection time, the alternate hypothesis was that FastGDP's performance was better. (i.e. lower). We set the threshold for statistical significance at $p < 0.05$.

Diagram Parsing Pipeline

FastGDP was written in the Python programming language (version 3.6.4). FastGDP uses the OpenCV Python library for image processing (13).

When performing primitive or point detection on a diagram image, FastGDP first uses connected component analysis as a pre-processing step (as text labels usually tend to be separate connected components) to remove all text labels from the image to prevent them from interfering with the line, circle, and corner detection **(Figure 4)**. To do this, the image is first binarized using Otsu's method (14) . This method is used as it avoids having to set a hardcoded threshold for binarizing the image. Then, the connected components in the input image are determined using OpenCV's connectedComponentsWithStats method, and the largest connected component is retained. All other connected components are replaced by white pixels.

After the preprocessing step, FastGDP detects circles in the diagram image using the Circle Hough Transform (CHT),

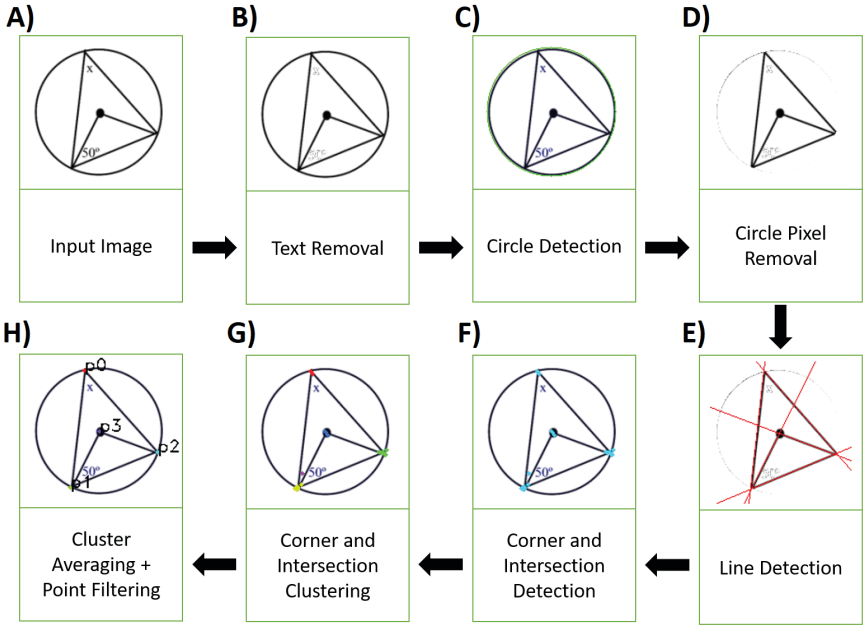

Figure 4: Summary of steps followed by FastGDP for point detection (primitive detection is an intermediate step) on a sample image. A) Input image. B) Result of text removal. C) Result of circle detection. D) Result of circle pixel removal. E) Result of line detection. F) Result of corner and intersection detection. G) Result of corner and intersection clustering. H) Result of cluster averaging and point filtering. Only steps B-E are used for primitive detection.

a standard circle detection algorithm **(Figure 4)** (3). The CHT is sensitive to the choice of parameters, especially the accumulator threshold. This threshold is the minimum value required in the accumulator array cell corresponding to a particular circle for the circle to be considered a true positive. If this threshold is too low, many false positive circles are detected, and if it is too high, no circles are detected even if the image contains circles. So, FastGDP uses binary search (for efficiency reasons) to determine the highest value of the accumulator threshold in the range [0, 150] for which at least one circle is detected. If this value is less than or equal to 70, FastGDP discards the results of the circle detector, since a very low value suggests that the image contains no circles, and the detected circles are false positives. At this stage, there could still be some duplicate circle detections, so FastGDP clusters the centers of the circles using the DBSCAN clustering algorithm and averages the center coordinates and radii within each cluster (10).

Next, FastGDP masks out all the circle pixels in the image using a disc-shaped mask for each circle **(Figure 4)**. This reduces the chance of false positive lines being detected along circles. Next, unlike GeoSolver, which uses the Standard Hough Transform, FastGDP uses the Progressive Probabilistic Hough Transform (PPHT) to detect lines **(Figure 4)** (3). The PPHT generally produces more accurate line detections, which circumvents the necessity of a primitive selection algorithm. However, the PPHT does sometimes detect duplicate lines. To overcome this, the lines are first converted to Hesse normal form since line endpoint data is not required for the subsequent steps. In Hesse normal form, a line is represented by two variables: rho (which represents the length of the perpendicular drawn from the origin to the line) and theta (which is the angle the perpendicular makes with the positive x axis). This also means that the dimensionality of the line data is reduced from four to two. Then, DBSCAN clustering is used in the rho-theta space followed by intra-cluster averaging. Since the rho-theta space is essentially in polar coordinates, the following metric is used instead of the standard Euclidean metric, where (ρ_1, θ_1) and (ρ_2, θ_2) are the rho-theta value pairs (in the Hesse normal form) for the first and second lines respectively and $d(\rho_1, \theta_1, \rho_2, \theta_2)$ is the value of the metric:

$$d(\rho_1, \theta_1, \rho_2, \theta_2) = \sqrt{\left(\frac{\rho_{diff}}{l_{max}}\right)^2 + \left(\frac{\theta_{diff}}{2\pi}\right)^2} \quad (6)$$

where

$$\rho_{diff} = |\rho_2 - \rho_1| \quad (7)$$

$$\theta_{diff} = min(|\theta_2 - \theta_1|, 2\pi - |\theta_2 - \theta_1|) \quad (8)$$

And ℓ_{max} is the maximum dimension of the image.

This metric is very similar to the one used by Liu, *et al.* (15).

At this stage, all primitives have been detected. While FastGDP does not need to detect line endpoints manually like GeoSolver, this can be done if necessary while incorporating FastGDP into a larger problem solver at minimal time overhead, since the final line detection usually does not contain more than 10 lines for most diagrams and usually contains 4-6 lines. In contrast, GeoSolver must detect endpoints of all over-generated lines. (By default, the number of over-generated lines is 40.)

Next, intersection points between primitives and corners are calculated **(Figure 4)**. For detecting corners, the Harris corner detector, a standard corner detection algorithm, is used (16). The output of the Harris corner detector is also used for removing false positive corner detections. For this purpose, the corner response map is first dilated generously to prevent correctly detected intersection points from being filtered out. Then only those intersection points are retained which are in a corner region.

The corners and intersection points are then clustered using the DBSCAN clustering algorithm **(Figure 4)**. The DBSCAN algorithm is used here (and during line and circle detection) mainly because the number of clusters is not known beforehand. For each cluster, the coordinates of the intersection points within the cluster, if the cluster contains at least one intersection point, or the corner points, if the cluster contains no intersection points, are averaged to determine the single point associated with that cluster. For each point so detected, FastGDP determines which (if any) detected lines or circles the point lines on. Finally, only those points are retained which lie on at least one line or circle or are the center of a circle. For images containing at least one circle, only those points are retained for which the corresponding cluster contains at least one intersection point, that is, clusters containing only corners are not retained **(Figure 4)**. This is because when circles are present in the image, the output of the Harris corner detector often contains many false positive corners around the circle. The points retained in the previous step are finally returned.

The source code for FastGDP with usage instructions can be found at bit.ly/FastGDPrepo. This GitHub repository also contains the three datasets mentioned in this paper with annotations for each dataset.

ACKNOWLEDGMENTS

We would sincerely like to thank the authors of the OpenCV library for providing excellent documentation and usage guides.

Received: November 25, 2021
Accepted: June 9, 2022
Published: October 2, 2022

REFERENCES

1. Chen, Xiaoyu, *et al.* "Automated generation of geometric theorems from images of diagrams." *Annals of Mathematics and Artificial Intelligence,* vol. 74, 2014, pp. 333-358.
2. Matas, J., *et al.* "Robust Detection of Lines Using the Progressive Probabilistic Hough Transform." *Computer Vision and Image Understanding,* vol. 78, no. 1, 2000, pp. 119–137. doi:10.1006/cviu.1999.0831.
3. Yuen, HK, *et al.* "Comparative Study of Hough Transform Methods for Circle Finding." *Image and Vision Computing,* vol. 8, no. 1, 1990, pp. 71–77. doi:10.1016/0262-8856(90)90059-e.
4. Seo, Minjoon, *et al.* "Solving Geometry Problems: Combining Text and Diagram Interpretation." *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics,* Sept. 2015, pp. 1466–1476.
5. Sachan, Mrinmaya and Eric Xing. "Learning to Solve Geometry Problems from Natural Language Demonstrations in Textbooks." *Proceedings of the 6th Joint Conference on Lexical and Computational Semantics (*SEM 2017), Association for Computational Linguistics,* Aug. 2017, pp. 251–261.
6. Lu, Pan, *et al.* "Inter-GPS: Interpretable Geometry Problem Solving with Formal Language and Symbolic Reasoning." *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), Association for Computational Linguistics,* Aug. 2021, pp. 6774– 6786.
7. Seo, Min Joon, *et al.* "Diagram Understanding in Geometry Questions." *Proceedings of the AAAI Conference on Artificial Intelligence,* vol. 28, no. 1, June 2014, doi:10.1609/aaai.v28i1.9146.
8. Chen, Jiaqi, *et al.* "GeoQA: A Geometric Question Answering Benchmark towards Multimodal Numerical Reasoning." *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021,* 2021, doi:10.18653/v1/2021.findings-acl.46.
9. Song, Dan, *et al.* "Retrieving Geometric Information from Images: The Case of Hand-Drawn Diagrams." *Data Mining and Knowledge Discovery,* vol. 31, no. 4, 2017, pp. 934–971., doi:10.1007/s10618-017-0494-1.
10. Ester, Martin, *et al.* "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise" *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, KDD'96,* 1996, pp. 226–23. doi: 10.5555/3001460.3001507
11. Bergstra, James, *et al.* "Algorithms for Hyper-Parameter Optimization." *Proceedings of the 24th International Conference on Neural Information Processing Systems,* 2011, pp. 2546–2554. doi: 10.5555/2986459.2986743
12. Sekachev, Boris, *et al.* "opencv/cvat: v1.6.0." Sep. 2021. doi:10.5281/zenodo.4009388.
13. Bradski, G. "The OpenCV Library." *Dr. Dobb's Journal of Software Tools*, 2000.
14. Otsu, Nobuyuki. "A Threshold Selection Method from Gray-Level Histograms." *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 9, no. 1, 1979, pp. 62–66., doi:10.1109/tsmc.1979.4310076.
15. Liu, Hui, *et al.* "Hough Transform and Clustering for a 3-D Building Reconstruction with Tomographic SAR Point Clouds." *Sensors,* vol. 19, no. 24, 2019. doi:10.3390/s19245378.
16. Harris, C., and M. Stephens. "A Combined Corner and Edge Detector." *Proceedings of the Alvey Vision Conference 1988,* 1988, doi:10.5244/c.2.23.