A Deep Reinforcement Learning Agent for Geometry Online Tutoring

Ziyang Xiao[1] and Dongxiang Zhang[1*]

[1]College of Computer Science and Technology, Zhejiang University, Hangzhou, Zhejiang, China.

*Corresponding author(s). E-mail(s): zhangdongxiang@zju.edu.cn;
Contributing authors: 22021206@zju.edu.cn;

Abstract

In this paper, we apply deep reinforcement learning (DRL) for geometry reasoning, and develop Dragon to facilitate online tutoring. Its success is contingent on a flexible data model to capture diverse concepts and heterogeneous relations, as well as an effective DRL agent to generate near-optimal and human-readable solutions. We use proximal policy optimization (PPO) as the backbone DRL architecture, customized with effective state representation and integrated with a bunch of optimization tricks including attention mechanism, action mask, data augmentation and curriculum learning. In our experimental study, we craft so far the largest scale dataset with geometry problems and a knowledge base with **46** theorems. We implement various heuristic algorithms and DRL models as baselines for performance comparison. The results show that our agent achieves near-optimal solution and is superior over multiple competitive baselines. To benifit the community, we opensource the dataset and implementation at https://github.com/AIEdu-xzy/geometry-solver.

Keywords: geometry problem, reinforcement learning, automatic reasoning

Statements and Declarations. No funding was received for conducting this study. The authors have no competing interests to declare that are relevant to the content of this article.

1 Introduction

Automatic geometry reasoning requires machines to conduct step-wise reasoning according to a prior knowledge base represented by a collection of theorems. It has attracted substantial attention from the communities of cognitive science and artificial intelligence and witnessed successful applications include automatic theorem discovery and proving [1], geometry problem synthesization [2] and smart tutoring systems [3, 4]. It is also helpful to measure the cutting-edge progress of machine intelligence in terms of the capability to leverage a prior knowledge base for automated logical deduction.

The early efforts devoted to automatic geometry theorem proving were from the community of mathematicians. In the pioneering work of [5], a novel algebra method was introduced, which for the first time proved hundreds of geometric theorems mechanically. Afterwards, various techniques such as Wu's method [6], Grobner basis method [7], and angle method [1] have been proposed. However, as remarked in [3], these approaches tend to produce arbitrary proofs in the underlying logical domain that are not friendly for non-experts in mathematics to understand. For instance, the idea proposed in Wu's method [6] is to first compute an irreducible zero decomposition and then determine the defining sets of the subvarieties from the irreducible ascending sets.

To generate readable solutions for a broader audience, especially for ordinary students or users of a tutoring system, Alvin *et al.* proposed the concept of hypergraph to facilitate automatic reasoning [2, 3]. Its nodes represent the geometric facts that are either explicitly provided or can be inferred by deduction. Each edge is of the form (F, f, t), indicating that a fact f can be deduced from a set of facts F, following theorem t. Reasoning is conducted by expansion from the source fact nodes, following the transition edges until the target node is reached. Since each reasoning step corresponds to a transition edge, the hypergraph model is more analogous to human-like thinking and demonstrates better readability. To find the most concise solution (i.e., with the minimum number of transition edges connecting the source fact nodes and the target node), a straightforward approach is to adopt BFS (breadth-first-search) to conduct step-wise expansion. In each iteration of expansion, all the theorems in the knowledge base are scanned to examine whether their premises can be satisfied by the current fact set F. Each applicable theorem generates a new fact node and a transition edge in the hypergraph. Even though such brute-force strategy guarantees to find the optimal solution, it becomes impractical when facing a complex geometry problem and a large knowledge base, since the search space grows exponentially with the number of reasoning steps and applicable theorems. To illustrate the idea, we plot in Figure 1 the number of new edges generated in each iteration of expansion, given a knowledge base with 46 theorems about triangle properties. We pick four geometry problems whose optimal solution contains 6, 7, 8 and 9 reasoning steps, respectively. In the early stage with a small number of reasoning steps, the size of hypergraph grows mildly because there are limited number of derived facts and applicable theorems. However, after a few reasoning steps, we can observe that the

size of hypergraph surges, leading to dramatically-increasing number of new fact nodes and more applicable theorems. For the problems with 9 reasoning steps (i.e., an optimal agent requires 9 transitions to solve the problems), BFS ultimately requires 4 million times of transition before the optimal solution is found and takes more than 30 hours to run in a commodity server.

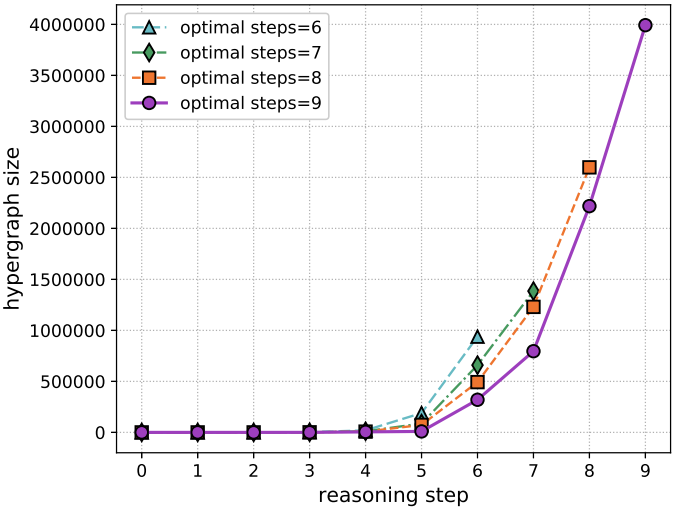


Fig. 1: Hypergraph size with increasing reasoning steps.

To sum up, existing geometry reasoning agents cannot achieve the goals of readability and efficiency at the same time. Furthermore, their extensibility to support large knowledge base and complex geometry problems remains unclear. To bridge the efficiency gap and support real-time tutoring systems in practice, we develop Dragon as the deep reinforcement learning (DRL) agent for geometry reasoning. Since geometry problems and theorems involve diverse entities, attributes and heterogeneous relations, Dragon relies on a flexible data model, analogous to RDF triples, to capture semantic relations. The problems and theorems are decoupled into sets of triplets. As to the learning model, we use proximal policy optimization (PPO) as the backbone DRL architecture, with customized state encoding and reward setting. Its performance is further boosted with multiple optimization tricks, including action mask to reduce action space, attention mechanism for better decision making, data augmentation for more training samples and curriculum learning for more effective learning strategy.

For performance evaluation, we craft a large-scale dataset with $1,000$ problems about triangle properties. The knowledge base contains 46 theorems. We also implement multiple heuristic algorithms and DRL models as the baseline solvers. Extensive experiments are conducted and the results show that our

proposed agent is superior over its competitors and can achieve near-optimal solutions in real time.

2 Related Work

In this section, we review relevant publications on automatic geometry problem solving and math reasoning.

2.1 Automatic Geometry Problem Solvers

As briefly reviewed in Introduction, automatic geometry reasoning has attracted attention from mathematicians decades ago [1, 5–7]. The solutions generated by machine are targeted for experts in the domain and lack readability for ordinary students. To improve readability, Alvin *et al.* proposed the concept of hypergraph in which each edge expansion corresponds to a reasoning step [2, 3]. However, its expansion strategy is deficient and cannot solve very complex problems. Another branch of works in automatic geometry problem solvers focuses on the parsing and alignment between problem text and geometry diagrams [8, 9]. For more details, readers can refer to a recent survey [10] on math word problem solvers. Notably, our research scope in this paper is focused on the component of automatic reasoning. Text and diagram parsing are complementary to our work and beyond the research scope.

2.2 DRL for Automatic Math Reasoning

There have been several deep learning based solvers proposed for automatic math reasoning. In [11], given a set of prior rules (e.g., $[\texttt{grandfatherOf}, X, Y]$:- $[[\texttt{fatherOf}, X, Z], [\texttt{parentOf}, Z, Y]]$), the objective is to conduct automatic reasoning for a query (e.g., $[\texttt{grandfatherOf}, Q, \texttt{BART}]$). Deep neural networks were introduced to learn vector representations of symbols and estimate the success score of a partial proof. In [12], Monte-Carlo search and reinforcement learning are applied to construct the proof tree of closed connection tableau for a set of clauses, which can be shown to be unsatisfiable if such a tree can be constructed [13]. NeuroSAT [14] predicts the satisfiability of SAT problems by training a binary classifier. The clauses and literals are represented as a bipartite graph and the input features are extracted via graph embedding, which are further encoded by multilayer perceptrons and two layer-norm LSTMs. The problem of automatic reasoning for Quantified Boolean formulas (QBF) was studied in [15]. Each formula is represented as a bipartite graph and encoded by graph neural networks for state representation. Policy network is then applied for action selection to pick the next available variable for reasoning.

Our differences: 1) Our geometry requires more complex state encoding for diverse entities, attributes and heterogeneous relations. 2) Instead of solving more problems with less time (as in these previous works), our objective is to

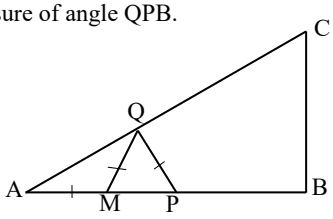
derive solutions as compact as possible. 3) We are the first to apply curriculum learning to improve math reasoning. 4) Dragon is integrated with more powerful optimization techniques.

3 Data Model

In this section, we briefly present the basic concepts in our data model, using Figure 2 to facilitate understanding, and the parsing of input problem text and theorems into logic representation.

Problem text:

ABC is a right triangle with the size of angle ACB equal to 74 degrees. The lengths of the sides AM, MQ and QP are all equal. Find the measure of angle QPB.



Knowledge base with theorems:

Theorem 1: If ABC is a triangle, then $\angle$ ABC + $\angle$ ACB + $\angle$ BAC = 180°

Theorem 2: If ABC is an isosceles triangle and AB=AC, then $\angle$ ABC = $\angle$ ACB

Theorem 3: Angles on one side of a straight line always add to 180 degrees

Fig. 2: An example of geometry problem and knowledge base.

- **Entity Type**. Geometry reasoning involves a diverse class of entity types including `Point`, `Segment`, `Angle`, `Triangle`, `Rectangle` and `Circle`, to name a few. Besides, there exist special variants of these entity types, such as `Isosceles_Triangle`, `Equilateral_Triangle`, and `Right_Triangle`.
- **Entity Instance**. Given a geometry problem, there exist multiple instances belonging to the same entity type. For example, we can construct more than 10 angle instances in Figure 2. It is also worth noting that arithmetic operators can be applied on the instances from the same type to generate new instances (e.g., $\angle AQP = \angle AQM + \angle PQM$).
- **Relation**. There exist heterogeneous relations between the entities. For instances, $\parallel$ and $\perp$ are common relations between two segments. `Centroid` describes the relation between a point and a triangle. To capture the relation between entity instances and their types, we also introduce `is` as a relation.

- **Triple**. Similar to the RDF triple to represent semantic relation between objects, we use $\langle o_i, r, o_j \rangle$ to capture the heterogeneous relation among entities, where o_i and o_j could be entity instance, entity types or constant values, and r represents the relation between o_i and o_j.
- **Fact**. We extract facts from the input geometry problem and represent them using triples. As an example, we can extract five triples from the problem in Figure 2: $\langle \triangle ABC, \texttt{is}, \texttt{Right\_Triangle} \rangle$, $\langle \angle ACB, =, 74 \rangle$, $\langle AM, =, MQ \rangle$, $\langle AM, =, QP \rangle$ and $\langle MQ, =, QP \rangle$.
- **Theorem**. In knowledge base, each theorem contains a set of conditions, known as hypotheses or premises, which can be used to deduce a conclusion. In our data model, we also use triples to represent the premises and conclusion in a theorem. The following presents our logic representation for the theorems in Figure 2:

$$\langle \triangle ABC, \texttt{is}, \texttt{Triangle} \rangle \Rightarrow \langle \angle ABC + \angle ACB + \angle BAC, =, 180 \rangle$$

$$\langle \triangle ABC, \texttt{is}, \texttt{Isosceles\_Triangle} \rangle \wedge \langle AB, =, AC \rangle \Rightarrow \langle \angle ABC, =, ACB \rangle$$

$$\langle C, \in, AB \rangle \Rightarrow \langle \angle ACD + \angle BCD, =, 180 \rangle$$

  In these examples of logic representation for theorems, the symbols $\{A, B, C, D\}$ are implemented as template slots and they can be instantiated to concrete entities. For example, it is possible for a triangle denoted by $\triangle MPQ$ to match the premise $\langle \triangle ABC, \texttt{is}, \texttt{Triangle} \rangle$ in Theorem 1, even though the symbols have no overlap.
- **Hypergraph**. We follow [2, 3] to use hypergraph for automatic reasoning. The nodes represent the explicit source facts extracted from the input problem and the deduced facts by applying theorems in the knowledge base. For each applicable theorem, we can find matching fact nodes for its premises and connect them to a new fact node generated from its conclusion.

3.1 Problem Text Parsing

It is challenging to construct an end-to-end model to support automatic parsing of geometry problems into logic representation. Existing methods confront a wide gap to work well in practice because as long as one fact node in the input problem is missing or mistakenly extracted, the solver fails to derive the correct solution. In this paper, we focus on explainable and efficient reasoning for (complex) geometry problems. The automatic parsing is beyond the scope and considered as our future work. We assume that human intervention is allowed to transform the input text and diagram into fact nodes.

To reduce the manual efforts, we propose a friendly input interface and a parsing algorithm with two steps to construct fact nodes. The first step is called *entity enumeration*, which accepts input in the format $\texttt{line}(p_1, p_2, \ldots, p_m)$, indicating that points p_1 to p_m are co-located in the same line. For example, we accept the following commands to model the geometry digram in Figure 2.

```
line(Q,M)  line(Q,P)  line(B,C)
line(A,Q,C)  line(A,M,P,B)
```

Our algorithm constructs the point entities mentioned in the input (i.e., points A, B, C, M, P, Q) and enumerates all the pairs (p_i, p_j) in $\texttt{line}(p_1, p_2, \ldots, p_m)$ as segment entities. The angles are derived from two input lines $\texttt{line}(p_1, p_2, \ldots, p_m)$ and $\texttt{line}(q_1, q_2, \ldots, q_m)$ sharing a common point. With these detected angles, we further construct all the possible triangles. After entity enumeration, the second step is *relation extraction*, which requires users to manually extract from the input problem. Our algorithm will build the fact nodes accordingly.

4 Deep Reinforcement Learning Agent

The network structure of our proposed DRL agent is illustrated in Figure 3. In the following, we present the details of each key component.

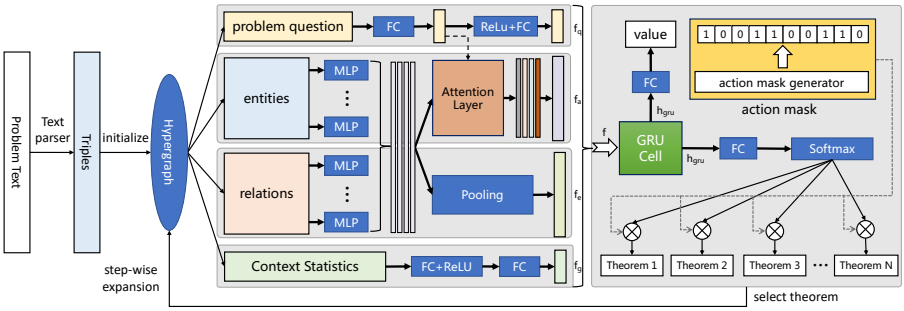


Fig. 3: The proposed DRL agent for geometry reasoning.

4.1 State Encoding

Since geometry reasoning is conducted on a hypergraph, a straightforward approach is to apply graph embedding for state encoding. Nevertheless, we observe that the edges in hypergraph represent the applicability of theorems and are not particularly helpful for state encoding. Thus, we propose a simple but effective encoding strategy to avoid the expensive computation overhead incurred by the embedding of dynamic graphs. Our state encoding is formed by the concatenation of the following features:

- Entity feature. We use one-hot encoding to represent entities. A pre-defined number of bits are allocated for all the possible entity types and attributes. A bit is set to 1 if the entity belongs to a particular entity type or the value on that attribute is known.
- Relation feature. Again, we use one-hot encoding to represent a relation.

- Question feature. The question in the input problem is the final target of reasoning and should be considered in state encoding to guide theorem selection. Its feature is generated by combining the embeddings of elements in the triplet representation.
- Context statistics feature. We also maintain various statistics as the context feature, such as the number of entities, relations, known angles, known segments, triangles with known sides, and triangles with known area.

The extracted features are further encoded by a fully-connected layer with ReLu activation, followed by another fully-connected layer with max pooling. We also use GRU cell to capture sequential dependency inherent in the order of reasoning steps.

4.2 Action Space

A geometry solution consists of multiple reasoning steps. In each step, we apply a theorem on a subset of fact nodes and generate a new fact node in the hypergraph. To construct a discrete action space with small size, we define an action as theorem selection from the knowledge base. It is possible that a theorem selected at a reasoning step matches multiple node sets in the current hypregraph. For example, the premise in Theorem 1 in Figure 2 matches all the triangle instances. To quickly identify the subsets of fact nodes that match the premises in a theorem, we build inverted index on the fly. We use entity types and relations as keys to insert the fact nodes. Given a theorem, we only need to examine the fact nodes with matching entity types and relations.

4.3 Reward Setting

We follow [15] to assign a small negative reward (-1 in our implementation) for each intermediate reasoning step to encourage the agent to solve the geometry problem with fewer steps. When a problem is solved successfully, we assign reward $+100$ to the last decision.

4.4 DRL Model

The agent is trained with proximal policy optimization (PPO) [16], which is considered as one of the state-of-the-art on-policy algorithms to learn a continuous or discrete control policy [17]. PPO defines a ratio $r_t(\theta)$ between the newly updated policy π_θ and old policy $\pi_{\theta_{\text{old}}}$ in the update step, i.e.,$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$. Its key contribution is to employ a clipped objective function to avoid large update to the model, so as to stabilize the training process:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\big[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\big]$$

Here, $\hat{A}_t$ is an estimator of the advantage function at timestamp t, and the expectation $\hat{\mathbb{E}}_t\big[\ldots\big]$ indicates the empirical average over a finite batch of samples. The loss function ensures that we make the maximum ratio of ϵ change to our policy at a time.

4.5 Optimization Tricks

Action Mask. Action mask is a popular technique for efficient exploration in discrete action space. It indicates the validity of an action for each state. In our application, if a theorem does not match any subset of the fact nodes, we disable this theorem selection from the action space.

Attention Mechanism. To enhance state encoding, we adopt attention mechanism which uses the problem question as the context to identify the importance weight of each entity and relation in the current state. Let $\boldsymbol{f_q}$ denote the feature vector of problem question and $\boldsymbol{h_{er}} = [\boldsymbol{f_e}, \boldsymbol{f_r}]$ be the concatenation of entity and relation features. Then, we use

$$\boldsymbol{\alpha} = \text{softmax}(\boldsymbol{f_q}^T \boldsymbol{W_a} \boldsymbol{h_{er}})$$

to derive the attention weight matrix, which is then applied on $\boldsymbol{h_{er}}$.

Data Augmentation. In computer vision, data augmentation is frequently used to enlarge the training set with the purpose of improving the diversity of samples and reducing overfitting [18–20]. Various operators, such as image translation, cropping, rotation, and shearing, have been investigated. In this paper, we propose three operators to synthesize new geometry problems and augment the training set with the solvable ones.

1. **Fact Replacement**: Given a solvable geometry problem, we randomly pick one fact node and change its value. The purpose is to render the agent pay less attention on the values, since it is highly probable that the optimal solution for the new problem is identical to the original problem.
2. **Fact Insertion**: The operator randomly picks and inserts a new fact node generated in the expansion of hypergraph by any existing solver. In other words, we manually inject redundancy into the fact node sets and expect the agent to be "smart" to only use a subset of fact nodes to generate the optimal solution.
3. **Question Replacement**: In this operator, we replace the question in the input problem to a new one, so as to increase the diversity of reasoning process.

Curriculum Learning. Curriculum learning [21] is inspired by the way humans acquire knowledge and has witnessed performance improvement in computation vision [22, 23] and natural language understanding [24]. Its model training starts from easy examples of a task and then gradually takes more difficult samples. In these applications, one of the key challenges is to define the extent of difficulty for the training samples. Fortunately, the difficulty of a geometry problem can be naturally measured by the number of reasoning steps

to solve it. Since BFS is infeasible to obtain the optimal solution for complex problems, we resort to our heuristic baselines, whose derived number of reasoning steps is viewed as an approximation to represent difficulty. Afterwards, we train the samples following an easy-to-difficult curriculum.

5 Experiments

5.1 Datasets

We craft a geometry dataset focused on triangle problems. It contains 300 problems and 46 theorems. From each problem, we can extract an average of 11.93 explicit fact nodes. The dataset covers 9 entity types and 14 relations.We set the split ratio for training, validation and test to 4:1:1. We use the 200 problems in the training set for data augmentation and synthesize 700 solvable problems as new training samples. We name these two datasets (w/ and w/o data augmentation) Tri-300 and Tri-1000, respectively. Note that they share the same validation and test sets.

5.2 Comparison Methods

For performance comparison, we implement a couple of competitive baselines, including four heuristic search algorithms, one hybrid approach with Monte Carlo tree search and RL, and three popular DRL models.

- **Bread-First Search (BFS)** maintains the intermediate results from all the applicable theorems in each reasoning step. It guarantees to find the optimal solution, but at the expense of enormous running time for complex problems.
- **Random Search (RS)** randomly selects an applicable theorem at each stage of expansion. The algorithm runs extremely fast but with very poor solution quality.
- **Beam Search (BS)** applies k theorems in each stage of expansion to control the size of hypergraph and can be viewed as a trade-off between BFS and RS. We $k = 100$ in our implementation.
- **Monte Carlo Tree Search (MCTS)** is a heuristic algorithm that figures out the best move from Monte Carlo simulations. It was applied in [12] for theorem proving.
- **MCTS+RL** was originally proposed for theorem proving [12]. It improves the selection stage in MCTS with reinforcement learning to learn more promising steps.
- **Policy Gradient (PG)** [25, 26] is a basic and popular type of DRL that relies upon optimizing parametrized policies by gradient descent.
- **Advantage Actor Critic (A2C)** [27] uses an advantage function instead of value function to improve training stability of the actor-critic networks. Compared with A3C (Asynchronous Advantage Actor-Critic), A2C was found to achieve the same or better performance in many tasks [16].

Method	Tri-300			Tri-1000		
	Steps	Memory	Time (s)	Steps	Memory	Time (s)
BFS	**5.4**	20.4 GB	1,787	**5.4**	20.4 GB	1,787
RS	12.7	27.9 MB	0.034	12.7	27.9 MB	0.034
BS	10.5	1.18 GB	11.7	10.5	1.18 GB	11.7
MCTS	6.9	166 MB	2.9	6.9	166 MB	2.9
MCTS+RL	6.4	131 MB	8.3	6.4	126 MB	8
PG	6.3	23.1 MB	0.801	6.1	22.3 MB	0.763
DDQN	6.5	27.6 MB	0.837	6.2	26.9 MB	0.783
A2C	6.4	26.9 MB	0.87	6.1	26.0 MB	0.806
Dragon	**6.1**	22.7 MB	0.771	**6**	21.5 MB	0.768

Table 1: Results on easy problems.

- **Double DQN (DDQN)** [28] reduces overestimations of action values by decomposing the max operation in traditional Q-learning into action selection and action evaluation, both of which are implemented with neural networks.

5.3 Parameter Setting

In the objective function of PPO, ϵ in the clip function is initialized to 0.3 and gradually decreases by 0.01 for every 1,000 episodes. In the training stage, we adopt dynamic learning rate using cosine annealing cycles [29], with the cycle length set to 1,000 episodes. The number of episodes, mini-batch size and the replay buffer capacity are set to 20,000, 100 and 2,000, respectively. All the experiments were conducted on the same server, with 16 CPU cores (Intel Xeon CPU E5-2650 with 2.30GHz), 256GB memory and NVIDIA GeForce GTX TITAN X.

5.4 Overall Performance

In the first experiment, we compare our approach with existing reinforcement learning models and heuristic algorithms in Tri-300 and Tri-1000. We set the maximum running time of BFS to 1 hour. A problem is called "easy" if it is solvable by BFS within the time budget. Otherwise, we categorize it as a "hard" problem. Among the 50 test problems, we obtain 19 easy problems and 31 hard problems. The baseline models of deep reinforcement learning, including PG, DDQN and A2C, are implemented with our proposed optimization techniques, such as attention mechanism, action mask and curriculum learning. At the end of the section, we will conduct a separate ablation study to evaluate the effect of these optimization tricks.

In Table 1, we report the average number of reasoning steps, memory consumption, and inference time for the easy problems. We can see that BFS achieves the optimal solution at extremely high computation cost. It takes 1,787 seconds to solve a relatively easy problem. Random search (RS) runs very fast, but generates poor solutions. Beam search (BS) slightly improves

Method	Tri-300			Tri-1000		
	Steps	Memory	Time (s)	Steps	Memory	Time (s)
RS	36.2	93.9 MB	0.109	36.2	93.9 MB	0.109
BS	32.4	5.95 GB	35.8	32.4	5.95 GB	35.8
MCTS	**22.3**	409 MB	26.7	**22.3**	409 MB	26.7
MCTS+RL	11.7	244 MB	78.5	10.3	244 MB	61.6
PG	8.6	43.2 MB	1.085	8.4	41.3 MB	1.052
DDQN	**8.4**	43.0 MB	1.077	8.1	38.5 MB	1.038
A2C	8.7	44.1 MB	1.189	8.5	41.8 MB	1.17
Dragon	**8.4**	41.9 MB	1.063	**7.9**	36.0 MB	1.021

Table 2: Results on hard problems.

RS, at the expense of more computation overhead. MCTS is the only heuristic algorithm that achieves satisfactory performance with reasonable running time. It is also worth noting that Tri-300 and Tri-1000 only differ in the size of training set. Since the heuristic methods require no training, their results are identical in the two test sets.

For the hybrid MCTS+RL, its solution quality is comparable to the DRL models (i.e., PG, DDQN, A2C and Dragon), and much better than vanilla MCTS, but with higher inference time. The reason is that it runs many Monto-Carlo simulations guided by reinforcement learning. It requires a neural network for action selection during the simulation and incurs more expensive computation cost.

As to the performance of DRL models, we have the following two observations. 1) Data augmentation plays a significant role for performance boosting. With extra diversified training samples, all DRL models enjoy noticeable improvement and achieve near-optimal number of reasoning steps. 2) Overall, PPO (implemented in Dragon) and DDQN demonstrate better performance than PG and A2C. Our explanation is that PG is the basic version of policy gradient and may suffer from the high variance in gradient estimation, which can be alleviated by PPO. In complex tasks, A2C may suffer from partial observability and long delays, which lead to unpromising performance.

For memory consumption, we can see that the DRL approaches, including PG, DDQN, A2C and Dragon, consume less amount of memory when compared with the other methods. The reason is that they can obtain the solution with fewer number of steps. More specifically, they explore smaller portion of nodes in the hypergraph before the target node is reached. Beam search (BS) algorithm consumes more memory than random search (RS) because it needs to maintain a huge number of intermediate results during the step-wise expansion.

The results on hard problems are shown in Table 2. We can see that data augmentation plays a more important role and causes deeper reduction of reasoning steps. Compared with beam search, the DRL models generate significantly better solutions with far less computation time. Our proposed Dragon also establishes clear superiority over the other competitors in these hard problems.

In Figure 4, we plot the training convergence of the DRL models in Tri-1000 with increasing number of episodes. The y-axis indicates the average reasoning steps for all the 50 test problems. The results show that policy gradient and its variants converge faster than DDQN in the early stage because they directly operate in the policy space. Nevertheless, with sufficient episodes, DDQN starts to outperformance PG and A2C in the task of geometric reasoning. PPO converges faster than the other competitors and achieves the best performance eventually.

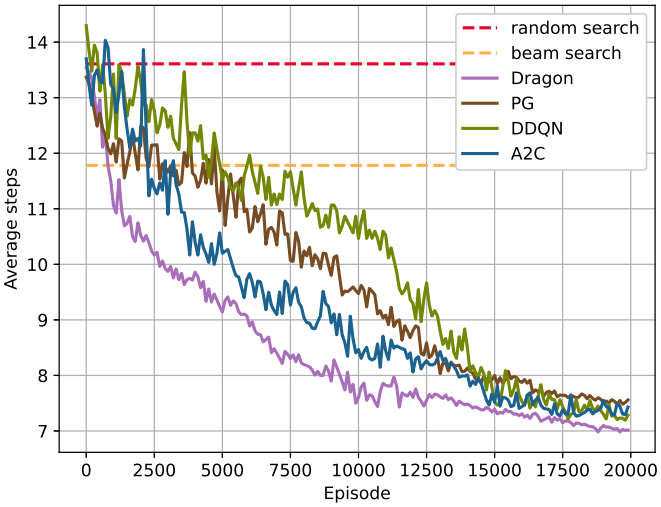


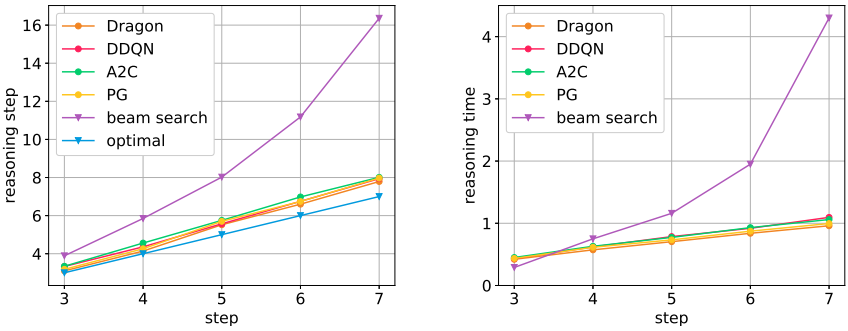
Fig. 4: Training convergence of DRL models with increasing number of episodes.

5.5 Scalability Analysis

We also examine the scalability of beam search and DRL models when handling problems with increasing number of steps in their optimal solutions. Figure 5 illustrates the solution quality and running time when the optimal number of steps increases from 3 to 7. We can see that the performance gap between beam search and DRL models is widened when the problems become more complex. The DRL models can always achieve near-optimal solutions and their running time increases linearly with the number of reasoning steps.

5.6 Ablation Study

In the last experiment, we conduct an ablation study to examine the effect of the proposed optimization techniques for our DRL agent, including action mask to reduce action space, GRU encoding and attention mechanism for



(a) Solution quality (b) Running time

Fig. 5: Solution quality and running time with increasing reasoning steps in optimal solutions.

better decision making in policy network, and curriculum learning to optimize the training strategy. From the results in Figure 6, we can see that action mask is a simple but very effective technique for performance boosting in DRL training. With fewer options in the action space, it brings the agent with much faster convergence in the early training stage. Curriculum learning also plays an important role, especially in the early stage of training. This is coherent with the way of human learning on math knowledge. The results verify that such easy-to-difficult curriculum is also useful for DRL agent. The attention mechanism and GRU cell are also helpful for decision making. They guide the training to select the correct answer and to certain extent, avoid random selection in the early training stage.

6 Conclusion

In this paper, we develop a deep reinforcement learning agent for automatic geometry reasoning. The model is able to generate near-optimal and human-readable solutions in real time, owning to the effective state encoding and a bunch of optimization tricks including action mask, data augmentation and curriculum learning. We craft a dataset with 300 triangle problems and augment it with 700 new training samples. Extensive experiments are conducted and the results verify the superiority of our agent over multiple heuristic algorithms and DRL baselines.

7 Acknowledgements

The work is supported by the National Key Research and Development Project of China (2022YFF0902000).

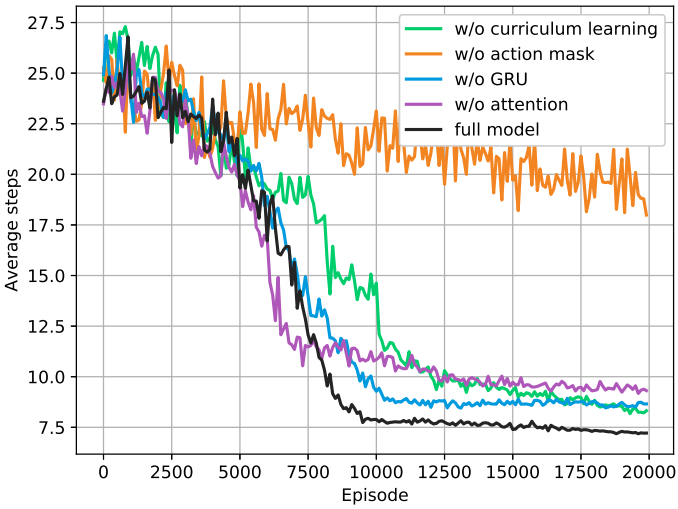


Fig. 6: Ablation study on the dataset of Tri-1000.

References

[1] Chou, S.-C., Gao, X.-S., Zhang, J.: Machine proofs in geometry: Automated production of readable proofs for geometry theorems (1994). https://doi.org/10.1142/9789812798152

[2] Alvin, C., Gulwani, S., Majumdar, R., Mukhopadhyay, S.: Synthesis of problems for shaded area geometry reasoning, pp. 455–458 (2017). https://doi.org/10.1007/978-3-319-61425-0_39

[3] Alvin, C., Gulwani, S., Majumdar, R., Mukhopadhyay, S.: Synthesis of geometry proof problems. Proceedings of the National Conference on Artificial Intelligence **1**, 245–252 (2014)

[4] Lan, Y., Wang, L., Zhang, Q., Lan, Y., Dai, B.T., Wang, Y., Zhang, D., Lim, E.: Mwptoolkit: An open-source framework for deep learning-based math word problem solvers. In: Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, pp. 13188–13190 (2022)

[5] TUN, W.-D.: On the decision problem and the mechanization of theorem-proving in elementary geometry. Scientia Sinica **21**(2), 159–172 (1978)

[6] Wen-Tsun, W.: Basic principles of mechanical theorem proving in elementary geometries. Journal of automated Reasoning **2**(3), 221–252 (1986)

[7] Kapur, D.: Using gröbner bases to reason about geometry problems.

Journal of Symbolic Computation **2**(4), 399–408 (1986)

[8] Seo, M.J., Hajishirzi, H., Farhadi, A., Etzioni, O.: Diagram understanding in geometry questions. In: AAAI, pp. 2831–2838 (2014)

[9] Seo, M.J., Hajishirzi, H., Farhadi, A., Etzioni, O., Malcolm, C.: Solving geometry problems: Combining text and diagram interpretation. In: EMNLP, pp. 1466–1476 (2015)

[10] Zhang, D., Wang, L., Zhang, L., Dai, B.T., Shen, H.T.: The gap of semantic parsing: A survey on automatic math word problem solvers. IEEE Trans. Pattern Anal. Mach. Intell. **42**(9), 2287–2305 (2020)

[11] Rocktäschel, T., Riedel, S.: End-to-end differentiable proving. In: NIPS, pp. 3788–3800 (2017)

[12] Kaliszyk, C., Urban, J., Michalewski, H., Olsák, M.: Reinforcement learning of theorem proving. In: NeurIPS, pp. 8836–8847 (2018)

[13] Letz, R., Mayr, K., Goller, C.: Cotrolled integration of the cut rule into connection tableaux calculi. J. Autom. Reason. **13**(3), 297–337 (1994)

[14] Selsam, D., Lamm, M., Bünz, B., Liang, P., de Moura, L., Dill, D.L.: Learning a SAT solver from single-bit supervision. In: ICLR (2019)

[15] Lederman, G., Rabe, M.N., Seshia, S., Lee, E.A.: Learning heuristics for quantified boolean formulas through reinforcement learning. In: ICLR (2020)

[16] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. CoRR **abs/1707.06347** (2017)

[17] Laskin, M., Lee, K., Stooke, A., Pinto, L., Abbeel, P., Srinivas, A.: Reinforcement learning with augmented data. CoRR **abs/2004.14990** (2020)

[18] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: NIPS, pp. 1106–1114 (2012)

[19] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: ICLR (2015)

[20] Cubuk, E.D., Zoph, B., Mane, D., Vasudevan, V., Le, Q.V.: Autoaugment: Learning augmentation strategies from data. In: CVPR, pp. 113–123 (2019)

[21] Bengio, Y., Louradour, J., Collobert, R., Weston, J.: Curriculum learning. In: Proceedings of the 26th Annual International Conference on Machine

Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009, pp. 41–48 (2009)

[22] Jiang, L., Zhou, Z., Leung, T., Li, L., Fei-Fei, L.: Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels. In: ICML, pp. 2309–2318 (2018)

[23] Guo, S., Huang, W., Zhang, H., Zhuang, C., Dong, D., Scott, M.R., Huang, D.: Curriculumnet: Weakly supervised learning from large-scale web images. CoRR **abs/1808.01097** (2018)

[24] Xu, B., Zhang, L., Mao, Z., Wang, Q., Xie, H., Zhang, Y.: Curriculum learning for natural language understanding. In: ACL, pp. 6095–6104 (2020)

[25] Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach. Learn. **8**, 229–256 (1992)

[26] Wu, S., Li, Y., Zhu, H., Zhao, J., Chen, G.: Dynamic index construction with deep reinforcement learning. Data Sci. Eng. **7**(2), 87–101 (2022)

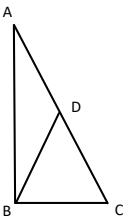
[27] Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T.P., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: ICML. JMLR Workshop and Conference Proceedings, vol. 48, pp. 1928–1937 (2016)

[28] Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: AAAI, pp. 2094–2100 (2016)

[29] Loshchilov, I., Hutter, F.: SGDR: stochastic gradient descent with warm restarts. In: ICLR (2017)

8 Appendix

Problem example 1



In triangle ABC, AB ⊥ BC, ∠BAC=30°,
∠BDC=60°, AD=1
What is the length of BC ?

Solution given by Dragon.

1. ∠DBC=60°, ∠BAC=30° ➡ ∠ABD=30° *[Exterior angle]*
2. AD=1, ∠BAC= ∠ABD ➡ DB=1 *[Law of sines]*
3. AB⊥BC ➡ ∠ABC = 90° *[Perpendicular lines]*
4. ∠ABC=90 °, ∠BAC=30° ➡ ∠ACB=60° *[Triangle angle sum]*
5. on △ABC ➡ BC = 1 *[Law of sines]*

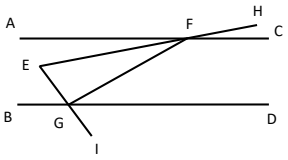
Solution given by MCTS.

1. AB⊥BC ➡ ∠ABC=90° *[Perpendicular lines]*
2. ∠ABC=90 °, ∠BAC=30° ➡ ∠ACB=60° *[Triangle angle sum]*
3. ∠ABC=90° ➡ ∠ABD=30° *[Angle sum]*
4. ∠BAC=30°, ∠ABD=30° ➡ ∠ADB=120° *[Triangle angle sum]*
5. ∠ADB=120°, AD=1, ∠ABD=30° ➡ AB=√3 *[Law of sines]*
6. ∠ABC=90°, AB=√3, ∠ACB=60° ➡ AC=2 *[Law of sines]*
7. ∠ABC=90°, AC=2, ∠BAC=30° ➡ BC=1 *[Law of sines]*

Solution given by MCTS-RL.

1. AB⊥BC ➡ ∠ABC=90° *[Perpendicular lines]*
2. ∠ABC=90 ➡ △ABC is RT *[Right triangle determination]*
3. ∠BDC=60° ➡ ∠ADB=120° *[Supplementary Angles]*
4. ∠ABC=90° ➡ ∠ABD=30° *[Angle sum]*
5. ∠ADB=120°, AD=1, ∠ABD=30° ➡ AB=√3 *[Law of sines]*
6. ∠ACB=90°, ∠BAC=30° ➡ ∠ACB=60° *[Triangle angle sum]*
7. ∠ACB=90°, AB=√3, ∠ACB=60° ➡ AC=2 *[Law of sines]*
8. △ABC is RT, AB=√3, AC=2 ➡ BC=1 *[Pythagorean theorem]*

Problem example 2



Lines AC and BD are parallel. ∠HFC=10°,
∠DGI=50°, △EFG is a right triangle,
and EG has a length of 10.
What is the length of EF?

Solution given by Dragon.

1. △EFG is RT ➡ ∠EGF=90° *[Right triangle property]*
2. ∠EGF=90°➡ ∠FGI=90 ° *[Supplementary Angles]*
3. ∠FGI=90°, ∠DGI=50°➡ ∠DGF=40° *[Angle sum]*
4. AC//BD ➡ ∠AFG=40° *[Parallel lines property]*
5. ∠CFH=10° ➡ ∠AFE=10° *[Vertical angles]*
6. ∠AFG=40°, ∠AFE=10° ➡ ∠EFG=30° *[Angle sum]*
7. ∠EGF=90°, EG=10, ∠EFG=30° ➡ EF=20 *[Law of sines]*

Solution given by MCTS.

1. ∠CFH=10° ⇒ ∠AFH=170° *[Supplementary Angles]*
2. ∠AFH=170° ⇒ ∠AFE=10° *[Supplementary Angles]*
3. △EFG is RT ⇒ ∠EGF=90° *[Right triangle property]*
4. ∠EGF=90° ⇒ ∠FGI=90° *[Supplementary Angles]*
5. ∠FGI=90°, ∠DGI=50° ⇒ ∠DGF=40° *[Angle sum]*
6. ∠DGF=40° ⇒ ∠CFG=140° *[Consecutive interior angles]*
7. ∠CFG=140°, ∠AFE=10° ⇒ ∠EFG=30° *[Angle sum]*
8. ∠EGF=90°, ∠EFG=30° ⇒ ∠FEG=60° *[Triangle angle sum]*
9. ∠EGF=90°, EG=10, ∠EFG=30° ⇒ EF = 10√3 *[Law of sines]*
10. △EFG is RT, EG=10, EF=10√3 ⇒ EF=20 *[Pythagorean theorem]*

Solution given by MCTS-RL.

1. ∠CFH=10° ⇒ ∠AFE=10° *[Vertical angles]*
2. △EFG is RT ⇒ ∠EGF=90° *[Right triangle property]*
3. ∠DGI=50° ⇒ ∠BGE=50° *[Vertical angles]*
4. ∠BGE=50° ⇒ ∠DGE=130° *[Supplementary Angles]*
5. ∠DGE=130°, ∠EGF=90° ⇒ ∠DGF=40° *[Angle sum]*
6. ∠DGF=40° ⇒ ∠CFG=140° *[Consecutive interior angles]*
7. ∠CFG=140° ⇒ ∠AFG=40° *[Supplementary Angles]*
8. ∠AFG=40°, ∠AFE=10° ⇒ ∠EFG=30° *[Angle sum]*
9. ∠EGF=90°, EG=10, ∠EFG=30° ➡ EF=20 *[Law of sines]*

Fig. 7: Examples of solutions generated by Dragon, MCTS and MCTS-RL.