Data Science and Engineering (2019) 4:179–192
https://doi.org/10.1007/s41019-019-00098-w

Towards Automatic Mathematical Exercise Solving

Tianyu Zhao[1] · Chengliang Chai[1] · Yuyu Luo[1] · Jianhua Feng[1] · Yan Huang[2] · Songfan Yang[2] · Haitao Yuan[1] · Haoda Li[1] · Kaiyu Li[1] · Fu Zhu[1] · Kang Pan[1]

Received: 24 June 2019 / Revised: 13 August 2019 / Accepted: 19 August 2019 / Published online: 6 September 2019

Abstract
Knowledge graphs are widely applied in many applications. Automatically solving mathematical exercises is also an interesting task which can be enhanced by knowledge reasoning. In this paper, we design MathGraph, a knowledge graph aiming to solve high school mathematical exercises. Since it requires fine-grained mathematical derivation and calculation of different mathematical objects, we design a crowdsourcing-based method to help build MathGraph. MathGraph supports massive kinds of mathematical objects, operations and constraints which may be involved in exercises. Furthermore, we propose an algorithm to align a semantically parsed exercise to MathGraph and figure out the answer automatically. Extensive experiments on real-world datasets verify the effectiveness of MathGraph.

Keywords Knowledge graph · Mathematical exercise · Knowledge reasoning · Crowdsourcing

1 Introduction

Currently, large-scale knowledge graphs are widely used in many real-world applications, such as semantic web search, question–answer systems, natural language processing and data analysis. For example, if we ask "What is the highest mountain?" on a web search engine, it may directly show the answer "Everest" with the help of a knowledge graph.

Recently, intelligent education has become more and more popular and automatically resolving mathematical exercises can help students improve the comprehensive ability. However, it is rather challenging to automatically resolve mathematical exercises without knowledge graphs, because it requires to use complex semantics and extra calculations. In this paper, we propose *MathGraph*, a knowledge graph aiming to solve high school mathematical exercises. MathGraph must be specially designed and differentiated from other knowledge graphs. The reasons are listed as follows:

1. *Knowledge in MathGraph belongs to a specific domain* Building MathGraph requires specific mathematical knowledge. Traditional knowledge graphs are built based on extensive semantic data, e.g. Wikipedia. However, it is very hard to get the semantic data for mathematical problems.
2. *Knowledge in MathGraph is stored in class level rather than instance level.* Most of the traditional knowledge graphs focus on extracting instances, categories and relations among instances. For example, a 3-tuple (Beijing, *isCaptialOf*, China) shows a relation between two instances. However, in MathGraph, there is no instance in the origin graph, but only many class-level mathematical objects (such as Complex Number and Ellipse). Only if an exercise is given, instances will be created accordingly.
3. *MathGraph supports mathematical derivation and calculation.* The reasoning process of mathematical problems is different from other problems, because besides logical relation, mathematical derivation must be included in the knowledge graph to solve mathematical exercises.

Moreover, there are numerous mathematical entities that need to be extracted, and it is very difficult to parse them automatically from the exercise texts. It is too expensive to ask a large number of experts to extract the entities for us. However, if we hire a few experts, it is difficult to derive complete entities in the domain. To address this, we decide to construct MathGraph via crowdsourcing. Existing works [4, 24] that focused on entity extraction of knowledge graph mainly extract entities from general web pages. However, our entities such as math

✉ Tianyu Zhao
zhaoty17@mails.tsinghua.edu.cn

[1] Tsinghua University, Beijing, China

[2] TAL Education Group, Beijing, China

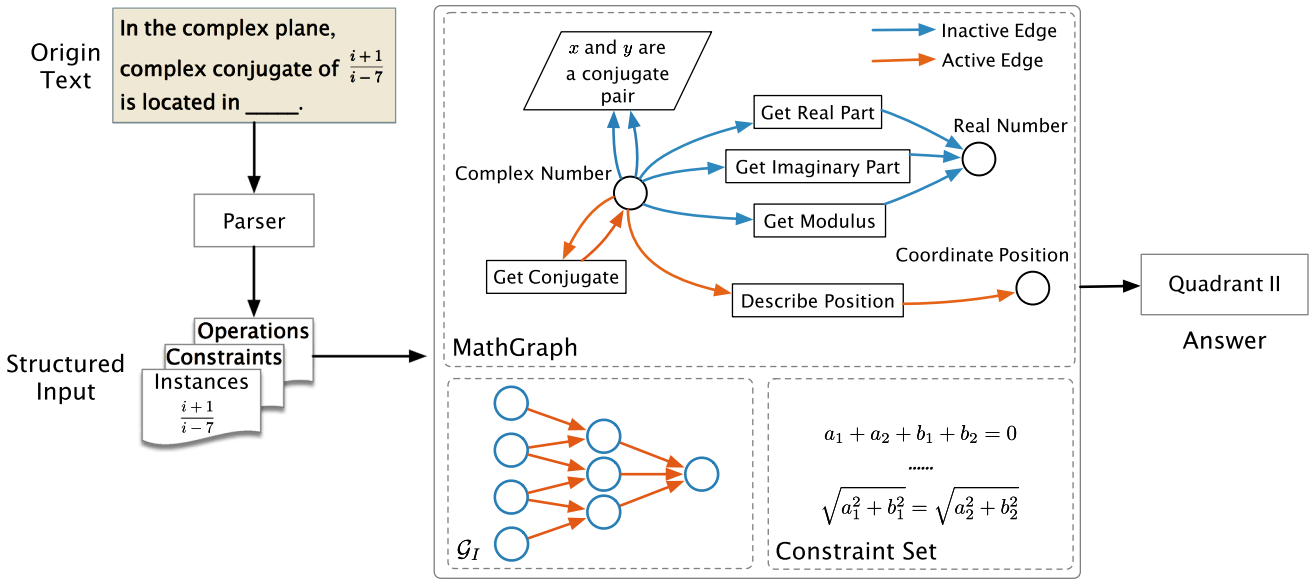


Fig. 1 Overview of using MathGraph to solve a mathematical exercise

objects, operations and constrains are from mathematical exercises, which is more complicated and domain-specific. Therefore, we have to design special tasks for MathGraph.

Thus, in this paper, we focus on designing and building a knowledge graph MathGraph for resolving mathematical problems. We also propose an effective algorithm to align a mathematical problem to MathGraph, and use the aligned sub-graph to resolve a mathematical exercise. Our contributions are as follows.

- We specially design the structure of MathGraph to support mathematical derivation and calculation. We model different mathematical objects, operations and constraints in MathGraph. To the best of our knowledge, this is the first attempt to build a knowledge graph for resolving mathematical problems.
- We propose an approach to construct MathGraph via crowdsourcing.
- We propose an algorithm to align a mathematical problem to MathGraph.
- We design a method to resolve mathematical exercises with the help of a semantic parser.
- Experimental study shows great performance of MathGraph and our proposed method.

Figure 1 gives an overview of the exercise-solving process with MathGraph. We detail the structure of MathGraph and the exercise-solving algorithm later.

The rest of this paper is organized as follows. Section 2 introduces some related works. Section 3 introduces some concepts involved in MathGraph. Section 4 overviews the structure of MathGraph. Section 5 introduces how to build MathGraph using crowdsourcing. Section 6 proposes some algorithms to solve mathematical exercises. Section 7 gives the experiment results, and we conclude the paper in Sect. 8.

2 Related Work

2.1 Reasoning with Knowledge Graph

Since knowledge graphs can provide well-structured information and relations of the entities, it is known to be useful to do reasoning in many tasks, such as query answering and relation inference (i.e. to infer missing relations in the knowledge graph [10, 21, 22]). Gu et al. [15] proposed a technique to answer queries on knowledge graph by "compositionalizing" a broad class of vector space models, which performs well on query answering and knowledge graph completion. Toutanova et al. [32] proposed a dynamic programming algorithm to incorporate all paths in knowledge graph within a bounded length, and modelled entities and relations in the compositional path representations. Zhang et al. [35] presented a deep learning architecture and a variational learning algorithm, which can handle noise in the question and do multi-hop reasoning in knowledge graph simultaneously. Zheng et al. [37] used a large number of binary templates rather than semantic parsers to query knowledge graph with natural language. A low-cost technique that can generate a large number of templates automatically is also proposed.

Our work is different from above works. Firstly, there are some differences between the structure of MathGraph and existing knowledge graphs (e.g. Freebase and NELL [3]). Secondly, to solve a math exercise usually requires multi-step

mathematical derivation, and the derivation procedures need to be output as the problem-solving process. Thirdly, derivation and calculation should be performed simultaneously when solving an exercise to retrieve the answer.

2.2 Automatically Solving Mathematical Problems

Automatically solving mathematical problems has been studied over years. But they only focused on easy problems, e.g. mathematical problems in primary schools. Kojiri et al. [18] constructed a mechanism called solution network to automatically generate the answers for mathematical exercises. The solution network is represented as a tree to describe inclusive relations of exercises. Tomas et al. [31] proposed a framework of constraint logic programming to automatically generate and solve mathematical exercises. This paper proposed to concentrate on the solving procedures rather than many simple exercise templates so that the generation and explanation of these exercises are easy. Ganesalingam et al. [13] proposed a method that solves elementary mathematical problems using logical derivation and shows solutions which are made difficult to distinguish from human's writing.

However, these works all have some limits. For example, some can solve those problems only involving elementary math (e.g. set theory, basic algebraic operation) without deeper theorems; some only support very limited logical derivation. Thus, in this paper, we present a knowledge graph to represent as many mathematical entities and logical relationships as possible.

2.3 Entity Extraction and Knowledge Graph Construction with Crowdsourcing

Crowdsourcing is widely used to extract entities and knowledge from massive types of data [5, 6, 14, 27, 34]. Chai et al. [4] focused on collecting entities using crowdsourcing with low cost and high quality. Dumitrache et al. [11] proposed a method for collecting medical relation using crowdsourcing. Seifert et al. [30] presented a method to extract entities from scientific literature, which further can be used to create an open knowledge base.

Crowdsourcing is also used to construct, update or integrate knowledge graph, such as Freebase [2]. Xin et al. [33] proposed a method for subjective knowledge base construction, which leverages crowd workers to annotate the subjective properties of the instances. McCoy et al. [23] used crowdsourcing to construct a clinical knowledge base by identifying relationships between medication pairs. Meng et al. [24] proposed a framework for large-scale knowledge base integration through crowdsourcing.

Compared with the conference version [36], we make the following contributions. Firstly, we design several user-friendly interfaces to leverage the crowd to build the MathGraph. Secondly, we design more quality control methods customized to the MathGraph construction problem. Thirdly, we conduct extensive experiment to evaluate the crowd-based method. Experiment results show that our method can achieve a higher quality than the expert-only approach while spending not so much money. Fourthly, we discuss more related works in this manuscript.

Table 1 Notations

Notation	Description
$\mathcal{G}$	MathGraph
v_o	An object node
v_p	An operation node
v_c	A constraint node
e_{DERIVE}	A DERIVE edge
e_{FLOW}	A FLOW edge
$\mathcal{O}$	A set of mathematical objects
$\mathcal{I}$	A set of mathematical instances
$\mathcal{C}$	A set of constraints
$\mathcal{G}_I$	A DAG describing dependency of all the uncertain instances

3 Preliminaries

In this section, we describe the entities that may appear in MathGraph, including mathematical objects and instances, operations and constraints. Table 1 shows the notations used in this paper.

3.1 Mathematical Object and Instance

A mathematical object is an abstract object which has a definition and some properties, and can be taken as the target of some operations or derivation. Note that a mathematical object can be defined in terms of other objects. A concrete object that satisfies the definition of the mathematical object is called an instance.

For example, COMPLEX NUMBER can be considered as a mathematical object:

- *Definition* A complex number is a number that can be in the form $a + bi$, where a and b are both real numbers and i is the imaginary unit which satisfies $i^2 = -1$.
- *Property example* Imaginary part is a property of a complex number. The imaginary part of a complex number $a + bi$ is b.
- *Operation example* $(a_1 + b_1 i) \cdot (a_2 + b_2 i) = (a_1 a_2 - b_1 b_2) + (a_1 b_2 + a_2 b_1) i$
- *Derivation example:* If $(a_1 + b_1 i)$ and $(a_2 + b_2 i)$ are conjugated to each other, then $a_1 = a_2$ and $b_1 + b_2 = 0$.

Table 2 Examples of key properties of different mathematical objects

Mathematical object	Example instance	Key properties
Complex number	$ai + b$	(a, b)
Elementary function	$f(x) = \langle$an algebraic expression about $x\rangle$	$\langle$The algebraic expression$\rangle$
Triangle	ΔABC	$(a, b, c, \angle A, \angle B, \angle C)$
Line	$Ax + By + C = 0$	(A, B, C)
Ellipse	$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$	(a, b)

And $2 + 3i$ and $(i + 1)(i - 3)$ are instances of Complex Number.

Different mathematical objects should be described as different structures in MathGraph. Thus, in MathGraph, a mathematical object is represented with a tuple of *key properties* $(p_1, p_2, \ldots, p_n)$. The key properties of a mathematical object are those properties that together can form and describe all the information of an instance of the object. Table 2 shows examples of key properties of some mathematical objects. Two instances of a mathematical object are equivalent if and only if all the key properties are equivalent.

In a mathematical exercise, instances can be categorized into certain instances and uncertain instances depending on whether it contains some uncertain values as its key properties. An instance is a *certain instance* if all key properties are certain, *uncertain instance* otherwise. For example, a real number 2.3 and a function $f(x) = x + \sin(x)$ are certain; a complex number $3 + ai$ (where $a \in \mathbb{R}$) and a random triangle ΔABC are uncertain.

3.2 Operation

Generally, an operation is an action or procedure which, given one or more mathematical objects as inputs (known as operands), produces a new object. Simple examples include addition, subtraction, multiplication, division and exponentiation. In addition, other procedures such as calculating the real part of a complex number, the derivative of a function and the area of a triangle can also be considered as operations.

3.3 Constraint

A constraint is a description or condition about one or more instances, at least one of which is an uncertain instance. There are four types of constraints: *descriptive constraints* (e.g. complex numbers x and y are conjugated), *equality constraints* (e.g. $a + 2 = b$), *inequality constraints* (e.g. $a^2 \leq 5$) and *set constraints* (e.g. $a \in \mathbb{N}$).

Most descriptive constraints cannot be applied directly to solve the exercise, but can be converted into other three types of constraints using some definitions or theorems. For example, if an exercise says "$a + 3i$ and $7 - bi$ are a conjugate pair", by the definition of conjugate complex, we can know that $a = 7$ and $3 + (-b) = 0$ by derivation.

4 The Structure of MathGraph

MathGraph is a directed graph $\mathcal{G} = \langle V, E\rangle$, in which each node $v \in V$ denotes a mathematical object, an operation or a constraint, and each edge $e \in E$ is the relation of two nodes.

4.1 Nodes

In general, nodes are categorized into three different types: *object nodes*, *operation nodes* and *constraint nodes*.

4.1.1 Object Nodes

An object node $v_o = (t, P, C)$ represents a mathematical object, where t denotes an instance template of this mathematical object; $P = (P_1, P_2, \ldots, P_n)$ is a tuple indicating key properties of the mathematical object; and C is a set of constraints that, according to the definition or some theorems, must be satisfied by this mathematical object. Table 3 shows an example of "triangle" as an object node. We can see that properties and theorems of triangles are included in the constraint set.

4.1.2 Operation Nodes

An operation node $v_p = (X_1, X_2, \ldots, X_k, Y, f)$ represents a k-ary operation, where $X_i (i = 1, 2, \ldots, k)$ and Y are object nodes representing the domain of the ith operand x_i and the result of the operation y, respectively, and f is a function that implements the operation and can be finished by a series of symbolic execution [1, 9, 17] process using a symbolic execution library (e.g. SymPy [26], Mathematica [16]) even if some operands are uncertain instances.

For example, *getting the modulus of a complex number* is an unary operation where $X_1 = \langle$`ComplexNumber`$\rangle$, $Y = \langle$`RealNumber`$\rangle$ and f can be implemented by the following symbolic execution process: (1) get the real part of x_1; (2) get the imaginary part of x_1; (3) return the squared root of the sum of (1) squared and (2) squared.

4.1.3 Constraint Nodes

A constraint node $v_c = (d, X_1, X_2, \ldots, X_k, f)$ represents a descriptive constraints of k instances, where d is the

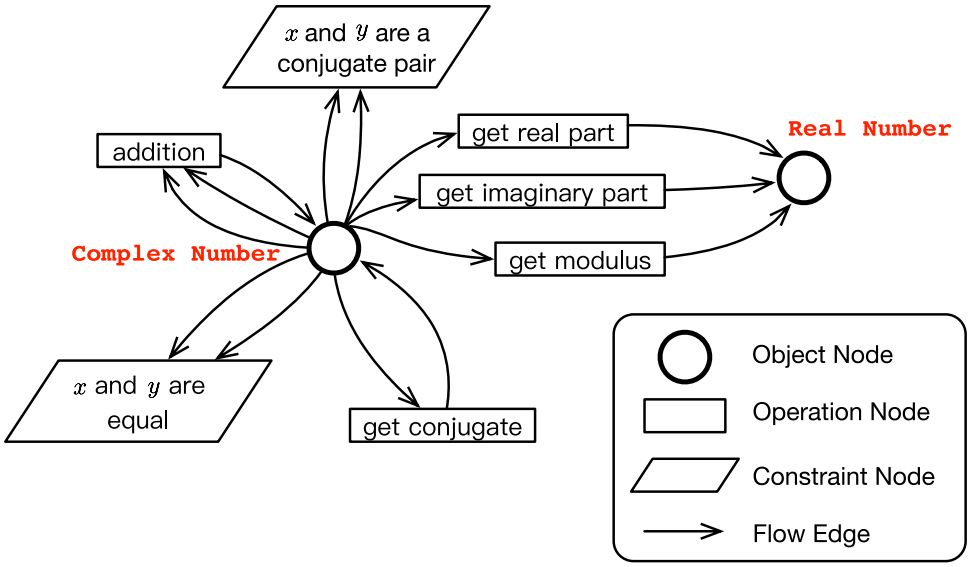


Fig. 2 Example of the FLOW edges

Table 3 An example of object node: triangle

Mathematical object	Triangle
Instance template	ΔABC
Key properties	(a, b, c, A, B, C)
Constraint set	$\{a, b, c > 0,$
	$0 < A, B, C < \pi,$
	$A + B + C = \pi,$
	$a + b > c, a + c > b, b + c > a,$
	$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C},$
	$a^2 = b^2 + c^2 - 2bc \sin A,$
	$b^2 = a^2 + c^2 - 2ac \sin B,$
	$c^2 = a^2 + b^2 - 2ab \sin C\}$

description of the constraint, $X_i (i = 1, 2, \ldots, k)$ are object nodes representing the domain of each involving instance, and f is a function which maps this descriptive constraint into several equality constraints, inequality constraints and set constraints.

For example, a constraint node represents that x_1 *and* x_2 *are a conjugate pair*, where $X_1 = X_2 = \langle$COMPLEX NUMBER$\rangle$ and f can be implemented by the following process: (1) get the real part of x_1 as a_1; (2) get the real part of x_2 as a_2; (3) get the imaginary part of x_1 as b_1; (4) get the imaginary part of x_2 as b_2; (5) return two equality constraints: $a_1 = a_2$ and $b_1 + b_2 = 0$.

4.2 Edges

There are two types of edges in MathGraph: the DERIVE edges and the FLOW edges.

4.2.1 The DERIVE Edge

For two object nodes X and Y, there may be a DERIVE edge $e_{\text{DERIVE}} = (X, Y, f)$ to indicate a general–special relationship between them, such as *Triangle* and *Isosceles Triangle*. If $X \xrightarrow{\text{DERIVE}} Y$, an instance of X can be reassigned as an instance of Y if certain conditions are met. These conditions are encapsulated into a function $f : X \to \{\text{False}, \text{True}\}$: if these conditions are met, the function f will return True and reassign the instance from X to Y; otherwise, it will simply return False.

For example, there is a DERIVE edge from object node TRIANGLE to ISOSCELES TRIANGLE, where the function f can be implemented as: (1) if the values of key properties or a constraint shows that two angles or lengths of two edges of the origin instance are equal, return an instance of *Isosceles Triangle* with the same key properties; (2) otherwise, return False.

When solving an exercise, reassigning an instance to a more specific object node will bring more constraints of this object and help find the answer. For example, for a rhombus $ABCD$, if we know that $\angle A = 90°$, we can infer, by the DERIVE edge from object node RHOMBUS to SQUARE, that $ABCD$ is a square and has constraints that $\angle A = \angle B = \angle C = \angle D = 90°$.

4.2.2 The FLOW Edge

A FLOW edge $e_{\text{FLOW}} = (X, Y)$ indicates the flow direction of instances during the exercise-solving process, which may only exist from an object node to an operation node, from an operation node to an object node or from an object node to a constraint node.

The FLOW edges between object nodes and operation nodes represent the process of passing instances as

parameters before the operation and the process of returning a new instance after it. For example, in Fig. 2, the two FLOW edges pointing to the operation node "addition" indicate that this operation takes two instances of complex number as its input values, and the edge leading from this operation node indicates that it returns a new instance of complex numbers.

The FLOW edges from object nodes to constraint nodes also represent the process of passing parameters of the constraints. For example, in Fig. 2, the two FLOW edges pointing to the constraint node "x and y are a conjugate pair" indicates that this constraint takes two complex number as its input. Note that constraints nodes only convert descriptive constraints into other types of constraints and generate no instances, so there are no FLOW edges from a constraint node to an object node.

In summary, MathGraph is a well-structured graph supporting different mathematical objects, operations and constraints. Next, we will discuss how to solve mathematical exercises using it.

5 MathGraph Construction using Crowdsourcing

As is mentioned above, MathGraph can be used to solve mathematical exercises. However, the objects, operations and constraints in MathGraph need to be extracted and refined by mathematical logic, so it is very difficult to construct MathGraph automatically. If one or a small group of people are chosen to create MathGraph manually, it is highly likely that some entities are missing, incomplete or incorrect. Therefore, we tackle this problem by leveraging the power of crowdsourcing to construct and validate the MathGraph.

Our whole task in this section can be described as follows. Given a set of mathematical exercises $\mathcal{R}$, we try to build MathGraph in a crowdsourcing platform (such as Amazon Mechanical Turk) and crowd workers. First of all, we need to extract all the mathematical objects, operations and constraints from the exercises. We randomly partition the exercise set $\mathcal{R}$ into several disjoint subsets $\{\mathcal{R}_1, \mathcal{R}_2, \dots\}$, where every subset contains no more than k ($1 \le k \le |\mathcal{R}|$) exercises. In practice, considering one worker can only handle limited exercises at once, k is recommended to take values between 5 and 20. Then we assign m crowd workers to each subset and design a set of user interfaces and questions to extract the objects, operations and constraints from text.

Quality control is also necessary in this task. Note that the workers who are familiar with the mathematical concepts and exercises can do a good job. Therefore, the hired workers need to know some fundamental and simple domain knowledge of math. To address this, we provide a detail instruction of the mathematical exercises which aims to guide the workers. In addition, to block the workers who are not qualified, we provide a quiz for each incoming worker. Only the workers who achieve high score can participate in the following tasks. Furthermore, since the answers still may contain incorrect or duplicated entities, we need to design corresponding algorithms to validate them.

5.1 Extracting Objects

The user interface we designed for extracting objects is shown in Fig. 3a. After workers submit their answers on the platform, we can obtain a collection of object names $\mathcal{O} = \{\mathcal{O}_1, \mathcal{O}_2, \dots, \mathcal{O}_m\}$, where $\mathcal{O}_i$ contains the names answered a worker. However, there exists two types of errors in the collected answers:

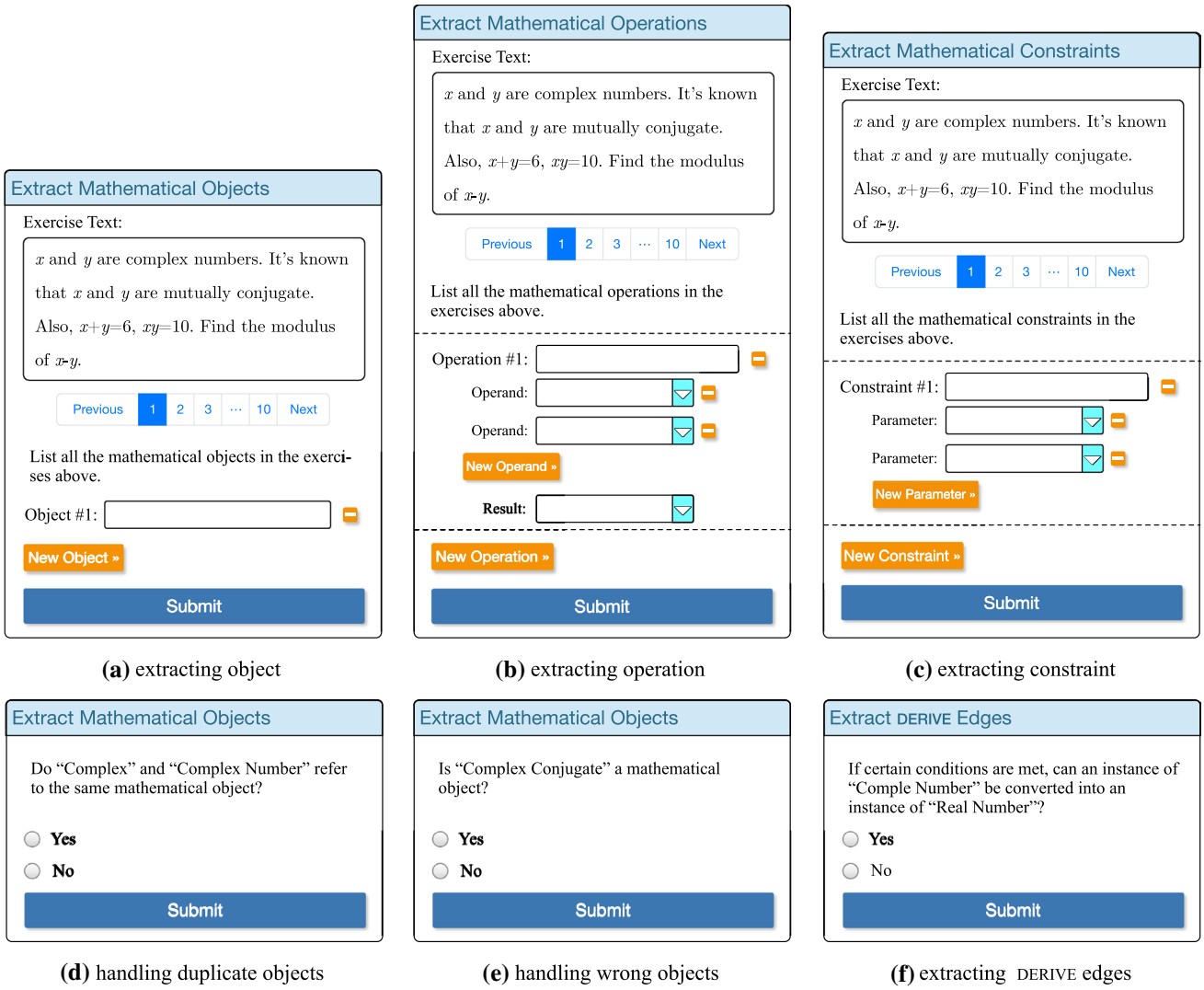
1. *Duplicate answers* Different names are actually referring to the same mathematical object, e.g. "Complex" and "Complex Number", "Point on the plane" and "Point in two-dimensional space".
2. *Wrong answers* Names of other entities are incorrectly categorized as mathematical objects. For instance, some worker submit "Complex Conjugate" as a mathematical object, which is actually an operation.

In order to solve the first type of error, we apply a classic entity resolution technique [7, 8]. Given a pair of collected object names $\{\{o_i, o_j\} | o_i, o_j \in \bigcup_{k=1}^{m} \mathcal{O}_k\}$, we can compute the similarity s_{ij} by utilizing any similarity function, e.g. Jaccard similarity, edit distance. We take Jaccard similarity as an example here. We first tokenize o_i into a set of tokens and compute Jaccard on token sets as follows.

$$s_{ij} = \text{JACCARD}(o_i, o_j) = \frac{|o_i \cap o_j|}{|o_i \cup o_j|}$$

Then, we select all pairs with similarity no less than a given similarity threshold (e.g. 0.3) and design questions for each pair to ask multiple workers whether two names are actually one object. Figure 3d shows an example of the question. After that, we can easily determine whether the pair should be merged into one object by these workers' (uniform or weighted [19]) majority vote.

As for the second type of error, we first count the number of occurrences of every name in $\mathcal{O}$, denoting as $c(o_i)$. The frequency of the name can be further defined as $f(o_i) = \frac{c(o_i)}{m}$. The higher the frequency, the more likely o_i is a mathematical object. Given a frequency threshold τ_f(e.g. 0.8), for a entity name o_i, (1) if $f(o_i) \ge \tau_f$, it will be inferred as a valid mathematical object; (2) if $f(o_i) \le 1 - \tau_f$, it will not be a mathematical object; (3) otherwise, we will transform it into a question (see Fig. 3e), send it to the crowdsourcing platform and obtain the answers from crowd workers.



(a) extracting object **(b)** extracting operation **(c)** extracting constraint

(d) handling duplicate objects **(e)** handling wrong objects **(f)** extracting DERIVE edges

Fig. 3 User interface designed for extracting entities

5.2 Extracting Operations and Constraints

Operations and constraints also need to be recognized and extracted from every given exercise subset. We also design the corresponding user interface on the crowdsourcing platform, shown in Fig. 3b, c. Note that the difference from extracting objects is that the workers have to submit not only the name of the operation/constraint, but also the type of the operands and the result of the operation, or the type of the parameters of the constraint. For example, the operation "Find the modulus of a complex number" should be submitted as a key-value map:

{
NAME: Complex Modulus,
OPERANDS: [Complex Number],
RESULT: Real Number
}.

Thus, we collect the workers' answer as sets: $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_m\}$ and $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m\}$, where every set $\mathcal{P}_i$ and $\mathcal{C}_i$ contains several operations and constraints, respectively. Here, an operation p is denoted as a tuple (p.name, p.operands, p.result) and a constraint c (c.name, p.parameters), where p.operands and c.parameters are both unordered list containing the domains of the operands of p and the parameters of c. Similar to the concept of type signature in programming languages, we define (p.operands, p.result) and c.parameters as the *signature* of p and c, respectively.

Extracting operations and constraints also face the same two possible errors as extracting objects. For the second type of error (i.e. a submitted entry is actually not an operation/constraint), we can follow the same approach as above. However, for the first type of error (i.e. one operation/constraint are submitted by several workers with different names), because the number of the operations and constraints in

MathGraph is much larger than the mathematical objects, and operations and constraints may have various description from different workers, simply using the same method above will result in too many questions for the workers. Therefore, a more efficient method is needed in this case.

We note that two different descriptions can refer to one operation/constraint only if their signatures are identical. Thus, we design Algorithm 1 to handle duplicated operations via crowdsourcing. First, we group these submitted entries based on the signature (line 2). Then for each entry in a group, we ask a crowdsourcing question to verify whether there is already an operation that has the same meaning (lines 11–19). If not, it will be considered as a new operation (lines 20–23).

Algorithm 1: HANDLINGDUPLICATEDOPERATIONS($\mathcal{P}$)

Input: $\mathcal{P}$: a set of submitted entries for operations.
Output: Θ: a set of set where each subset containing submitted entries referring to the same operations.

```
1  begin
2      Group all elements in P by their signatures. All
         the groups are stored in Σ = {σ_1, σ_2, ···};
3      Initialize Θ as empty result set;
4      foreach σ_i ∈ σ do
5          Initialize Ω as an empty list of list;
6          Randomly choose an entry ê from σ_i;
7          Initialize ω as an empty list;
8          ω.push(ê);
9          Ω.push(ω);
10         foreach e ∈ σ_i \ {ê} do
11             flag = True;
12             for i = 1 to Ω.length do
13                 e' = a random entry in Ω[i];
14                 Design question q = "Do e and e'
                     refer to the same operations?";
15                 Upload q to the platform and retrieve
                     workers' vote as v;
16                 if v == "yes" then
17                     Ω[i].push(e);
18                     flag = False;
19                     break;
20             if flag then
21                 Initialize ω as an empty list;
22                 ω.push(e);
23                 Ω.push(ω);
24         Θ = Θ ∪ Ω;
25     return Θ
```

5.3 Extracting Edges

To construct a complete MathGraph, we still need to extract the edges. The FLOW edges can be created automatically by the submitted signature of the operations and constraints. For instance, according to the signature of the operation "Complex Modulus", two FLOW edges (Complex Number, Complex Modulus) and (Complex Modulus, Real Number) will be added into MathGraph.

However, the DERIVE edges contain extra information, so they have to be extracted by crowdsourcing. For every ordered pair of the mathematical objects, we design a question as shown in Fig. 3f to ask the workers whether there is a FLOW edge between these two objects, and retrieve the answer from multiple workers' majority vote.

After extracting all the nodes and edges in MathGraph through crowdsourcing, several experts (i.e. people who can write code) are asked to program the logical information in these nodes and edges, such as the function f in an operation node v_p (see Sect. 4.1.2). At last, we can construct MathGraph from extracted entities by crowds and coded logic by experts.

6 Solving Mathematical Exercises with MathGraph

In this section, we propose a framework to solve a mathematical exercise using MathGraph. First, we use a semantic parser mapping exercise text to the instances, operations and constraints, respectively. Then, we solve the constraints and update uncertain instances. Finally, we return the answer of this exercise.

6.1 Mapping Text in MathGraph

Considering the limited information and expression in the mathematical exercises, we can easily use a rule-based semantic parser to parse the exercise text and then map them to corresponding nodes in MathGraph.

The rule-based semantic parser uses a set of rules to parse every sentence of the exercise and recognize the logical relationship in the text. For example, "Let x and y be complex numbers" will be parsed as declaration of two uncertain instances; "Find the coordinates of the conjugate complex of $(i+1)(i-1)$" will be parsed as a declaration of a certain instance and two operations.

6.1.1 Mapping Instances

With the semantic parser, every instance generated from the exercise should have already mapped into the corresponding object node. That is, a set of instances $\mathcal{I} = \{(x_1, X_1), \ldots, (x_k, X_k)\}$ is generated by parsing the text of the exercise, where x_i denotes the instance and X_i denotes the corresponding object node.

Instances are classified as certain instances or uncertain instances depending on whether the exercise provide certain values or expressions of them. For uncertain instances generated from text, key properties with unknown value should be generated as instances, since they may be used in the operations and constraints of this exercise. For example, for the exercise shown in Fig. 4, x and y are both uncertain instances

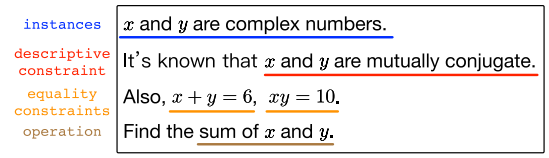


Fig. 4 Parsing the text into nodes in MathGraph

of object node COMPLEX NUMBER. Therefore, we need to generate a_x, b_x, a_y and b_y as four uncertain instances of object node REAL NUMBER, where a_x and b_x stand for the two key properties of x, and a_y and b_y stand for the key properties of y.

6.1.2 Mapping Operations

The semantic parser can also parse out the a set of operations from the text. Every operation in it will be aligned to the corresponding operation node in MathGraph with its operands, trigger the function in the operation node and then finally generate a new instance as the output of the operation.

Algorithm 2: MAPPINGTEXT$(t, \mathcal{G})$

Input: t: text of the exercise;
$\mathcal{G}$: MathGraph
Output: $\mathcal{I}_{\text{certain}}$: a set of certain instances;
$\mathcal{I}_{\text{uncertain}}$: a set of uncertain instances;
$\mathcal{C}$: a set of constraints;
$\mathcal{S}_{\text{dependency}}$: a set denoting dependencies of uncertain instances;

1 **begin**
2 Initialize P as a semantic parser;
3 $\mathcal{I}_{\text{certain}}, \mathcal{I}_{\text{uncertain}} \leftarrow$ P.MAPPINGINSTANCES$(t, \mathcal{G})$;
4 $\mathcal{O} \leftarrow P$.MAPPINGOPERATIONS$(t, \mathcal{G})$;
5 $\mathcal{C} \leftarrow P$.MAPPINGCONSTRAINTS(t, G);
6 Let $\mathcal{S}_{\text{dependency}}$ be an empty set;
7 **for** each $(x, X) \in \mathcal{I}_{uncertain}$ **do**
8 **for** each key property$(p, X_p) \in x.keyProperties$ **do**
9 **if** p is an uncertain instance **then**
10 $\mathcal{I}_{\text{uncertain}} \leftarrow \mathcal{I}_{\text{uncertain}} \cup \{(p, X_p)\}$;
11 $\mathcal{S}_{\text{dependency}} \leftarrow \mathcal{S}_{\text{dependency}} \cup \{(p, x)\}$;
12 **for** each $(o, (x_1, X_1), \cdots, (x_k, X_k)) \in \mathcal{O}$ **do**
13 $(y, Y) = o.f(x_1, \cdots, x_k)$;
14 **if** y is a certain instance **then**
15 $\mathcal{I}_{\text{certain}} \leftarrow \mathcal{I}_{\text{uncertain}} \cup \{(y, Y)\}$;
16 **else**
17 $\mathcal{I}_{\text{uncertain}} \leftarrow \mathcal{I}_{\text{uncertain}} \cup \{(y, Y)\}$;
18 $C \leftarrow S \cup y.ConstraintSet$;
19 **for** $i = 1$ **to** k **do**
20 **if** x_i is an uncertain instance **then**
21 $\mathcal{S}_{\text{dependency}} \leftarrow \mathcal{S}_{\text{dependency}} \cup \{(x_i, y)\}$;
22 **for** each $(c, (x_1, X_1), \cdots, (x_k, X_k)) \in \mathcal{C}$ **do**
23 **if** c is a descriptive constraint **then**
24 $c \leftarrow c.f(x_1, \cdots, x_k)$;
25 **return** $\mathcal{I}_{certain}, \mathcal{I}_{uncertain}, \mathcal{C}, \mathcal{S}_{dependency}$

6.1.3 Mapping Constraints

Similar to mapping operations, for every descriptive constraint $(c, (x_1, X_1), \ldots, (x_n, X_n))$ in the exercise, the semantic parser can map it to the corresponding constraint node c with some involving instances, trigger the function in the node and convert it to several equality/inequality/set constraints.

Also, note that when an uncertain instance is generated, some constraints may also be generated according to the constraint set of the corresponding object node. After that, we gather all the constraints in the exercise as a set for further using.

Algorithm 2 shows the process of mapping text of the exercise, where instances are mapped in lines 7–11, operations are mapped in lines 12–21, and constraints are mapped in lines 22–25.

6.2 Solving Uncertain Instances and Constraints

After parsing all the instances and operations in the exercise, the answer of the exercise should already be generated as an instance (from the text or by an operation). If this instance is a certain instance, we can directly return the value of this instance as the answer; otherwise, we must deal with these uncertain instances and solve the constraints in the exercise to update their values and finally retrieve the answer of the exercise.

6.2.1 Reassign Uncertain Instances

First, we need to check every uncertain instance whether it can be reassigned to a more specific object node in MathGraph by a DERIVE edge. For an uncertain instance i that is assigned to an object node v_o, we check every outgoing DERIVE edge of v_o, and if the function of an edge e returns true, then we reassign i to the object node that e points to and add all the constraints in this node to the constraint set. Algorithm 3 shows the pseudocode of this process.

For example, if we have an uncertain instance ΔABC, and there is a constraint $\angle B = \angle C$ in the constraint set, then the DERIVE edge from TRIANGLE to ISOSCELES TRIANGLE should return true. So the instance should be reassigned to ISOSCELES TRIANGLE, and a new constraint $AB = AC$ should be added to the constraint set.

Algorithm 3: REASSIGNINSTANCES$(\mathcal{G}, \mathcal{I}_{\text{uncertain}}, \mathcal{C})$

Input: $\mathcal{G}$: MathGraph;
$\mathcal{I}_{\text{uncertain}}$: the set of uncertain instances;
$\mathcal{C}$: the constraint set;

1 **begin**
2 **for** each instance $(x, X) \in \mathcal{I}_{\text{uncertain}}$ **do**
3 **for** each DERIVE edge $(X_e, Y_e, f_e) \in \mathcal{G}$ **do**
4 **if** $X_e == X$ **and** $f_e(x) == True$ **then**
5 $\mathcal{C} \leftarrow \mathcal{C} \cup Y_e.ConstraintSet$;
6 update (x, X) as (x, Y);

6.2.2 Organizing Uncertain Instances

Note that for two uncertain instances α and β, there may be a dependency relationship between them, which is caused because either α is one of the inputs of an operation node and β is the output or α is one of the key properties of β.

Thus, we use a graph $\mathcal{G}_I = \langle V_I, E_I \rangle$ to describe dependency of all the uncertain instances, where $v \in V_I$ is a node representing an uncertain instance and $e \in E_I$ is a directed edge representing a dependency relationship of two nodes. Note that $\mathcal{G}_I$ is always a DAG, since there will be no dependency loop in it.

Let $\mathcal{S}_I = \{v | v \in V_I \wedge \forall u \in V_I, (u, v) \notin E_I\}$ denote the set containing all node without any incoming edges in $\mathcal{G}_I$. It is obvious that if all nodes in $\mathcal{S}_I$ can turn into certain instances, the instance corresponding to the answer can be derived to a certain instance. Algorithm 4 demonstrates this process.

Algorithm 4: OrganizeInstances($\mathcal{I}_{\text{uncertain}}, \mathcal{S}_{\text{dependency}}$)

Input: $\mathcal{I}_{\text{uncertain}}$: a set of uncertain instances;
$\mathcal{S}_{\text{dependency}}$: the set denoting dependencies of uncertain instances;
Output: $\mathcal{G}_I$: the graph to organize the uncertain instances;
$\mathcal{S}_I$: a set denoting all instances in $\mathcal{G}_I$ without incoming edges;

```
begin
    Let G_I⟨V_I, E_I⟩ be an empty graph;
    for (x, y) ∈ S_dependency do
        V_I ← V_I ∪ {x, y};
        E_I ← E_I ∪ {(x, y)};
    S_I ← {v | v ∈ V_I ∧ ∀u ∈ V_I, (u, v) ∉ E_I};
    return G_I, S_I
```

For example, Fig. 5 shows $\mathcal{G}_I$ of the exercise in Fig. 4, where x and y depend on their respective key properties, and $z = x + y$ depends on its two operands. In this context, $\mathcal{S}_I = \{a_x, b_x, a_y, b_y\}$ and the instance corresponding to the answer is z.

6.2.3 Organizing and Solving Constraints

After the last step, we now have a set of constraints. First, we need to make sure every variable in every constraint is in $\mathcal{S}_I$. If not, this constraint needs to be rewritten by using its key properties as the variable. For example, for the exercise in Fig. 4, the set of the constraint is $\{x + y = 6, xy = 10, a_x = a_y, b_x + b_y = 0\}$. Since $x, y \notin \mathcal{S}_I$, the first two constraints will be rewritten as $a_x + b_x i + a_y + b_y i = 6$ and $(a_x + b_x i)(a_y + b_y i) = 10$.

Now the constraint set includes and formalizes all the constraints in the exercise. So we can apply methods of a symbolic execution library [16, 26] or some approximation algorithms [12, 29] to solve these equations and/or inequalities. Finally, we will get the value (or range of value) of every instance in $\mathcal{S}_I$. Algorithm 5 shows this process.

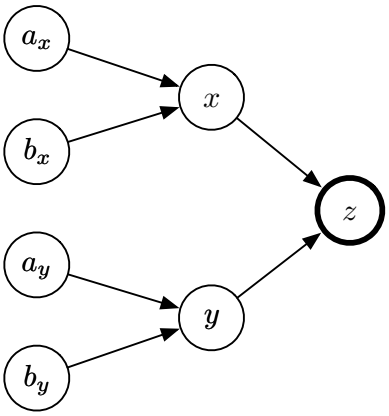


Fig. 5 $\mathcal{G}_I$: A DAG to organize the uncertain instances

Algorithm 5: ProcessConstraints($\mathcal{C}, \mathcal{G}_I, \mathcal{S}_I$)

Input: $\mathcal{C}$: the constraint set;
$\mathcal{G}_I$: the graph for dependency of uncertain instances;
$\mathcal{S}_I$: the set denoting all instances in $\mathcal{G}_I$ without incoming edges;

```
begin
    for each (c, (x_1, X_1), ··· , (x_k, X_k)) ∈ C do
        for i = 1 to k do
            if x_i ∉ S_I then
                Replace (x_i, X_i) with its key
                  properties (p_1, P_1), ··· , (p_n, P_n);
    SolveConstraints(S_constraint, S_I);
```

6.2.4 Updating Uncertain Instances and Retrieving the Answer

After solving all the constraints in the exercise, we need to update the value of the rest instances in $\mathcal{G}_I$. Since we now know the value of instances in $\mathcal{S}_i$, we can traverse every instance in $\mathcal{G}_I$ in the topological sorting order and update their values in turn. Finally, we return the value of the instance corresponding to the answer. Algorithm 6 shows the complete process of using MathGraph to solve exercise.

Algorithm 6: SolvingExercise($t, \mathcal{G}$)

Input: t: text of the exercise;
$\mathcal{G}$: MathGraph
Output: answer of the exercise

```
begin
    I_certain, I_uncertain, C, S_dependency ←
      MappingText(t, G);
    Mark the instance corresponding to the answer
      as x_ans;
    ReassignUncertainInstances(G, I_uncertain, C);
    G_I, S_I ←
      OrganizeUncertainInstances(I_uncertain, S_dependency);
    ProcessConstraints(C, S_I);
    Update the value of every node in G_I in the
      topological sorting order;
    return value of x_ans;
```

7 Experiments

In this section, we conduct extensive experiments on real mathematical datasets to evaluate the performance of our method.

7.1 Datasets and Experiment Setting

We collect four real-world datasets of mathematical exercises of Chinese high schools, namely Complex, Triangle, Conic and Solid. The exercises are stored in plain text, and the mathematical expressions are stored in the LaTeX format.

- Complex This dataset contains 1526 mathematical exercises related to calculation and derivation of complex numbers, including basic algebraic operation, the modulus and the conjugate of a complex number, Argand plane, polar representation, etc.
- Triangle This dataset contains 782 mathematical exercises related to solving triangles (using law of sines and law of cosines), which includes finding missing sides and angles, perimeter, area, radius of the circumscribed circle, etc.
- Conic This dataset contains 1196 exercises related to Conic sections, including calculation and derivation on ellipse, hyperbola and parabola.
- Solid This dataset contains 653 exercises related to solid geometry, which involves a variety of geometries in three-dimensional Euclidean space, including pyramids, prisms, etc.

Exercises in the four datasets are categorized into three levels (i.e. easy, medium and hard) based on the difficulty (which is classified according to the accuracy of many high school students). Table 4 shows the number of exercises with different difficulty levels in the datasets.

In the experiments, we use Neo4j [28] as the graph database platform to build and index MathGraph. For the datasets, we build the knowledge graph manually involving only the instances, operations and constraints that may exist in these exercises. All algorithms are implemented in Python 3.7. Sympy [25] is used to do the work of symbolic execution. All the experiments are conducted in a machine with 2.40 GHz Intel Xeon CPU E52630, 48 GB RAM, running Ubuntu 14.04.

7.2 MathGraph Construction

We randomly choose 50% exercises from each dataset and use them to construct MathGraph. Those elements to be extracted are done by workers on ChinaCrowds[1] [20], which is a user-friendly crowdsourcing platform. In the experiment, we set $m = 5$. Moreover, for task Fig. 3a–c, we pay 5 RMB each. For each task Fig. 3d–f, we pay 1 RMB because they are simpler than the above tasks. The total cost is 15,480 RMB. We compare with the baseline that utilizes 3 experts to do extraction on a sampled dataset (due to the limited ability of a single expert) on precision, recall and F1-score. The ground truth is retrieved by multiple experts proofreading the crowdsourcing result.

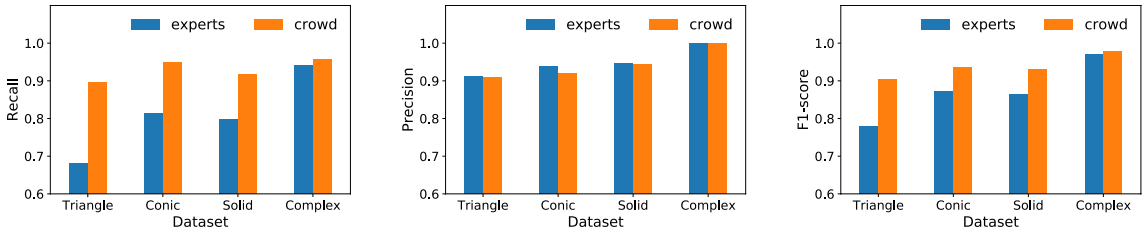
Table 4 Summary of exercises in the datasets

	Easy	Medium	Hard	Total
Complex	685	634	207	1526
Triangle	179	470	133	782
Conic	486	602	108	1196
Solid	217	336	100	653

As shown in Fig. 6a, for all datasets, our crowd-based strategy has a much higher recall than the expert-based strategy because we use 5 workers to answer a task and combine their answers. For the expert-based strategy, each expert has to answer a lot of questions, and thus, they cannot cover so many entities, which results in a low recall. For example, on Triangle dataset, crowd-based strategy has a recall of 89%, which is 20% more than that of expert-based strategy (68%). On Complex dataset, crowd-based and expert-based strategies achieve similar recalls (95% and 94%, respectively) because the dataset is simple and has a small number of entities to be extracted. Therefore, experts can also achieve a high recall.

For precision, as shown in Fig. 6b, the expert-based strategy can achieve a high precision because it leverages the human's expertise. However, we can see that our crowd-based strategy is comparable with the expert-based one because we remove the duplicated answers and verify the wrong answers. For example, on Complex dataset, both methods have a precision of 100%. Moreover, on Conic dataset, crowd-based and expert-based strategies achieve a similar precision of 93% and 92%, respectively. Overall, for the F1-score, Fig. 6c shows that crowd-based method is better than the expert-based one because our method has a much higher recall and a comparable precision compare with the expert-based solution. For example, on Solid dataset, crowd-based strategy has a recall of 93%, which is better than that of expert-based strategy by 7%.

[1] http://www.chinacrowds.com/.



(a) Recall **(b)** Precision **(c)** F1-score

Fig. 6 Evaluation of the crowd-based entity extraction strategy

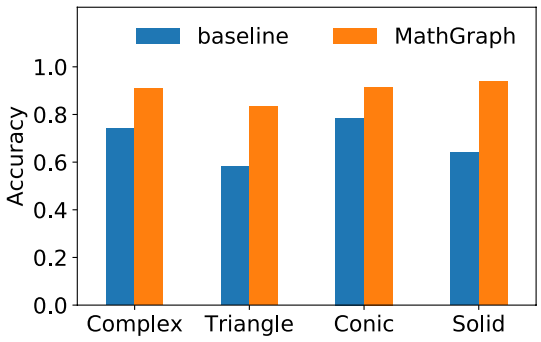


Fig. 7 Overall accuracy on four datasets

7.3 Exercise Solving

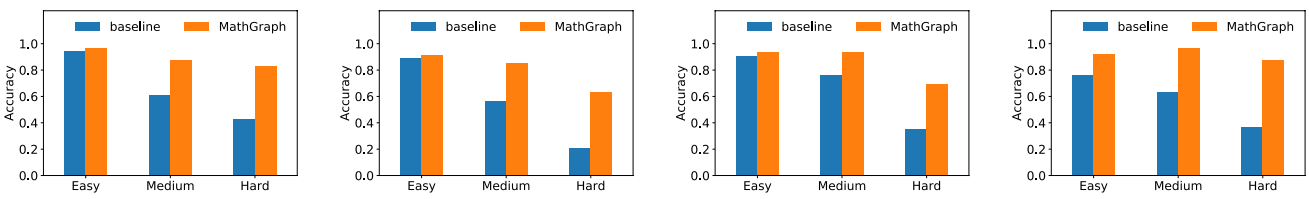
We implement a rule-based baseline method as the following procedures:

1. We still use a rule-based semantic parser to parse the text and extract the information.
2. A large quantity of rules are written in advance to match different situations of exercises. We randomly selected 20% exercises and assign 8 programmers to program rules, which can align and solve the exercises in this set. Every rule represents an exercise type and has a built-in solving process only for this exercise type.
3. Then, these rules are used to solve all exercises. If an exercise matches a rule, then we apply the solving process of the rule and return the answer.

MathGraph is created by crowdsourcing and proofread by experts. It only includes nodes and edges of the four types of the exercises in our dataset, containing 89 mathematical objects, 723 operations and 875 constraints. Figure 7 shows the exercise-solving accuracy on four datasets. We can see that in every dataset, our method achieves higher accuracy than baseline, e.g. 20% higher accuracy. This result shows the effectiveness of solving problems using MathGraph.

Figure 8 demonstrates the exercise-solving accuracy on different difficulty levels. From the experiment result, we have the following observations. Firstly, as the difficulty of the exercises increases, the accuracy of both methods decreases. Secondly, for easy exercises, the baseline and our method have similar performance, but for medium and hard exercises, MathGraph significantly outperforms the baseline, because our method can use the knowledge graph to do mathematical derivation.

The rule-based baseline considers the exercise as a whole and solving it according to the logic specified by a rule. This means that this method relies on a large amount of rules, and the more complex the exercise is, the more rules and the higher difficult it needs to write. Therefore, this method has a poor performance in hard exercises. However, our method extracts the mathematical objects, calculations and constraints from these rules and models them into a graph, so it can be used for multi-step calculation and derivation.



(a) Dataset `Complex` **(b)** Dataset `Triangle` **(c)** Dataset `Conic` **(d)** Dataset `Solid`

Fig. 8 Accuracy on different difficulty levels

8 Conclusion

In this paper, we proposed MathGraph, a knowledge graph for automatically solving mathematical exercises. MathGraph is specially designed to represent different mathematical objects, operations and constraints. Considering the complexity of the semantics of the mathematical exercises, we use crowdsourcing to construct MathGraph. Given an exercise, we can use the proposed method to solve it with the help of MathGraph and a pre-built semantic parser. Experimental study on four real-world datasets demonstrates the accuracy of our method.

References

1. Baldoni R, Coppa E, D'Elia DC, Demetrescu C, Finocchi I (2018) A survey of symbolic execution techniques. ACM Comput Surv 51(3):50
2. Bollacker KD, Evans C, Paritosh P, Sturge T, Taylor J (2008) Freebase: a collaboratively created graph database for structuring human knowledge. In: Proceedings of the ACM SIGMOD international conference on management of data, SIGMOD 2008, Vancouver, BC, Canada, June 10–12, 2008, pp 1247–1250
3. Carlson A, Betteridge J, Kisiel B, Settles B, Hruschka Jr E. R, Mitchell T. M (2010) Toward an architecture for never-ending language learning. In: Proceedings of the twenty-fourth conference on artificial intelligence (AAAI 2010), vol 5, Atlanta, p 3
4. Chai C, Fan J, Li G (2018) Incentive-based entity collection using crowdsourcing. In: 34th IEEE international conference on data engineering, ICDE 2018, Paris, France, April 16–19, 2018, pp 341–352
5. Chai C, Fan J, Li G, Wang J, Zheng Y (2018) Crowd-powered data mining. CoRR, abs/1806.04968
6. Chai C, Fan J, Li G, Wang J, Zheng Y (2019) Crowdsourcing database systems: overview and challenges. In: 35th IEEE international conference on data engineering, ICDE 2019, Macao, China, April 8–11, 2019, pp 2052–2055
7. Chai C, Li G, Li J, Deng D, Feng J (2016) Cost-effective crowdsourced entity resolution: a partial-order approach. In: Proceedings of the 2016 international conference on management of data, SIGMOD conference 2016, San Francisco, CA, USA, June 26–July 01, 2016, pp 969–984
8. Chai C, Li G, Li J, Deng D, Feng J (2018) A partial-order-based framework for cost-effective crowdsourced entity resolution. VLDB J 27(6):745–770
9. Cousot P, Cousot R (1977) Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, ACM, pp 238–252
10. Dongo I, Cardinale Y, Chbeir R (2018) Rdf-f: Rdf datatype inferring framework. Data Sci. Eng. 3(2):115–135
11. Dumitrache A, Aroyo L, Welty C (2018) Crowdsourcing ground truth for medical relation extraction. TiiS 8(2):11:1–11:20
12. Fletcher R, Leyffer S (2003) Filter-type algorithms for solving systems of algebraic equations and inequalities. In: High performance algorithms and software for nonlinear optimization, Springer, pp 265–284
13. Ganesalingam M, Gowers WT (2017) A fully automatic theorem prover with human-style output. J Autom Reason 58(2):253–291
14. Gao Y, Miao X (2018) Query processing over incomplete databases. Synthesis lectures on data management. Morgan & Claypool Publishers, San Rafael
15. Guu K, Miller J, Liang P (2015) Traversing knowledge graphs in vector space. In: Proceedings of the 2015 conference on empirical methods in natural language processing, EMNLP 2015, Lisbon, Portugal, September 17–21, 2015, pp 318–327
16. Inc WR (2018) Mathematica, Version 11.3. Champaign, IL
17. King JC (1976) Symbolic execution and program testing. Commun ACM 19(7):385–394
18. Kojiri T, Hosono S, Watanabe T (2005) Automatic generation of answers using solution network for mathematical exercises. In: International conference on knowledge-based and intelligent information and engineering systems, Springer, pp 1303–1309
19. Li G, Chai C, Fan J, Weng X, Li J, Zheng Y, Li Y, Yu X, Zhang X, Yuan H (2017) CDB: optimizing queries with crowd-based selections and joins. In: Proceedings of the 2017 ACM international conference on management of data, SIGMOD conference 2017, Chicago, IL, USA, May 14–19, 2017, pp 1463–1478
20. Li G, Chai C, Fan J, Weng X, Li J, Zheng Y, Li Y, Yu X, Zhang X, Yuan H (2018) CDB: a crowd-powered database system. PVLDB 11(12):1926–1929
21. Li K, Li G (2018) Approximate query processing: What is new and where to go? Data Sci Eng 3(4):379–397
22. Lin P, Song Q, Wu Y (2018) Fact checking in knowledge graphs with ontological subgraph patterns. Data Sci Eng 3(4):341–358
23. McCoy AB, Wright A, Laxmisan A, Ottosen MJ, McCoy JA, Butten D, Sittig DF (2012) Development and evaluation of a crowdsourcing methodology for knowledge base construction: identifying relationships between clinical problems and medications. JAMIA 19(5):713–718
24. Meng R, Chen L, Tong Y, Zhang CJ (2017) Knowledge base semantic integration using crowdsourcing. IEEE Trans Knowl Data Eng 29(5):1087–1100
25. Meurer A, Smith CP, Paprocki M, Čertík O, Kirpichev SB, Rocklin M, Kumar A, Ivanov S, Moore JK, Singh S et al (2017) Sympy: symbolic computing in python. PeerJ Comput Sci 3:e103
26. Meurer A, Smith CP, Paprocki M, Čertík O, Kirpichev SB, Rocklin M, Kumar A, Ivanov S, Moore JK, Singh S, Rathnayake T, Vig S, Granger BE, Muller RP, Bonazzi F, Gupta H, Vats S, Johansson F, Pedregosa F, Curry MJ, Terrel AR, Roučka v, Saboo A, Fernando I, Kulal S, Cimrman R, Scopatz A (2017) Sympy: symbolic computing in python. PeerJ Comput Sci 3:e103
27. Miao X, Gao Y, Guo S, Liu W (2018) Incomplete data management: a survey. Front Comput Sci 12(1):4–25
28. Neo4j I. Neo4j, Version 1.1.12. https://neo4j.com/
29. Polyak BT (1964) Gradient methods for solving equations and inequalities. USSR Comput Math Math Phys 4(6):17–32
30. Seifert C, Granitzer M, Höfler P, Mutlu B, Sabol V, Schlegel K, Bayerl S, Stegmaier F, Zwicklbauer S, Kern R (2013) Crowdsourcing fact extraction from scientific literature. In: Human–computer interaction and knowledge discovery in complex, unstructured, big data: third international workshop, HCI-KDD 2013, Held at SouthCHI 2013, Maribor, Slovenia, July 1–3, 2013. Proceedings, pp 160–172
31. Tomás AP, Leal JP (2003) A clp-based tool for computer aided generation and solving of maths exercises. In: International

symposium on practical aspects of declarative languages, Springer, pp 223–240
32. Toutanova K, Lin V, Yih W.-t, Poon H, Quirk C (2016) Compositional learning of embeddings for relation paths in knowledge base and text. In: Proceedings of the 54th annual meeting of the association for computational linguistics, vol 1, pp 1434–1444
33. Xin H, Meng R, Chen L (2018) Subjective knowledge base construction powered by crowdsourcing and knowledge base. In: Proceedings of the 2018 international conference on management of data, SIGMOD conference 2018, Houston, TX, USA, June 10–15, 2018, pp 1349–1361
34. Yan Q, Huang H, Gao Y, Ying C, Hu Q, Qian T, He Q (2016) Modeling for noisy labels of crowd workers. In: 18th Asia-Pacific web conference on APWeb 2016Web technologies and applications, Suzhou, China, September 23–25, 2016. Proceedings, part II, pp 227–238
35. Zhang Y, Dai H, Kozareva Z, Smola AJ, Song L (2018) Variational reasoning for question answering with knowledge graph. In: Proceedings of the thirty-second AAAI conference on artificial intelligence (AAAI-18), pp 6069–6076
36. Zhao T, Huang Y, Yang S, Luo Y, Feng J, Wang Y, Yuan H, Pan K, Li K, Li H, et al (2019) Mathgraph: a knowledge graph for automatically solving mathematical exercises. In: International conference on database systems for advanced applications, Springer, pp 760–776
37. Zheng W, Yu JX, Zou L, Cheng H (2018) Question answering over knowledge graphs: question understanding via template decomposition. Proc VLDB Endow 11(11):1373–1386