Geometric Search in TGTP

Yannis Haralambous, Pedro Quaresma

Geometric Search in *TGTP*

Yannis Haralambous[1], Pedro Quaresma[2]

[1] IMT Atlantique Bretagne-Pays de la Loire, Computer Science Department
UMR CNRS 6285 Lab-STICC, Technopôle Brest-Iroise CS 83818,
29238 Brest Cedex 3, France
yannis.haralambous@imt-atlantique.fr
[2] CISUC / Department of Mathematics, University of Coimbra
Apartado 3008, EC Santa Cruz, 3001-501 Coimbra, Portugal
pedro@mat.uc.pt

Abstract. In this age of information the importance of retrieve the knowledge from the many sources of information is paramount. In Geometry, apart from textual approaches, common to other areas of mathematics, there is also the need for a geometric search approach, i.e. semantic searching in a corpus of geometric constructions.
The Web-based repository of geometric problems Thousands of Geometric problems for geometric Theorem Provers (*TGTP*) has, from the start, some text search mechanisms. Since version 2.0 an implementation of the geometric search mechanism is integrated in it. Using a dynamic geometry system it is possible to build a geometric construction and then semantically search in the corpus for geometric constructions that are super-sets of the former, with regard to geometric properties.
It should be noted that this is a semantic check, the selected constructions may not look like the query construction, but they will possess similar sets of geometric properties.

Keywords: Geometric automated theorem proving · Repository of geometric problems · Common formats · Geometric search · Conceptual graphs · Typed sub-graph isomorphism

1 Introduction

Having repositories of information, one of the first question to solve is how to browse the information contained within. Regarding repositories of geometric information we should add to the usual text searches, geometric searches, i.e. we should be able to provide a geometric construction and look for similar constructions.

Searching the *TGTP* repository can be done in three ways: a simple textual query, a more comprehensive textual search, and a geometric search [7].

The simple textual query is done using *MySQL* regular expressions queries [6], over the `name` attribute of the `Conjectures` table, it will provide the list of conjectures with names similar to the query. Another, more powerful, textual

query mechanism is available, using the *full-text search* of *MySQL* [6]. The attributes `name, description, shortDescription, keyword` of the `theorems` and `keywords` tables are used, allowing, for a given input sentence, to get the list of most similar sentences in any attribute of the different problem descriptions.

Based on some preliminary work on geometric search [4] we developed a geometric search mechanism. The queries are constructed using a dynamic geometry system (*GeoGebra*[3]) and the constructed figure is semantically compared with the figures in the repository.

2 Geometric Search in *TGTP*

2.1 Conceptual Graphs

Knowledge representation provides techniques for describing objects in a knowledge domain, using concepts and relations defined by consensus in a community of users. In the case of Euclidean geometry the choice of concepts and relations is quite straightforward: we will use points, segments, lines (see 2.2). Once the signature decided, there are several mathematical structures for building knowledge bases. We have chosen to use *conceptual graphs* [1] rather than OWL ontologies based on RDF triples because the former allow relations of arbitrary arity and because conceptual graphs can be processed with graph theory algorithms.

To give an example, the (trivial) geometric figure of a single line segment AB is represented by a CG of four concepts and three relations (see Fig. 1): two concepts of type point, with markers A and B, one concept of type segment, with marker S_{AB} and one concept of type line, with marker L_{AB}; the three relations are: between A and S_{AB} as well as between B and S_{AB} there are binary relations I "is incident to," and between S_{AB} and L_{AB} there is a binary relation C "contained in":

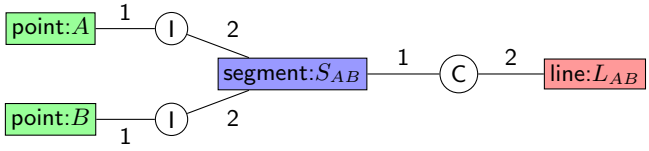

Fig. 1. Conceptual Graph, Single Line Segment AB

The semantics are as follows: point:A and point:B are distinct[4] extremities of segment:S_{AB}, which, in turn, is contained in line:L_{AB}. When converting a

[3] `https://www.geogebra.org/`

[4] We consider that every concept of the graph represents a distinct geometric object. Whenever inference reveals that two concepts represent the same object, they are merged.

geometric figure from some other representation to conceptual graphs, geometric constraints of the figure (as in [5, §2.2]) become conceptual graph relations. Furthermore, geometric inference rules become conceptual graph λ-rules. We repeatedly apply inference rules until inferential closure is obtained. By doing this a search will succeed in finding a figure even if it has been originally described in a different but geometric equivalent way. For example, if a figure has been converted into a conceptual graph as a *triangle with three equal sides* and the user searches a *triangle with three equal angles* (which is geometrically equivalent), the search will be successful because—thanks to inferential closure—the property that angles are equal will be already part of the graph (see Fig. 2).

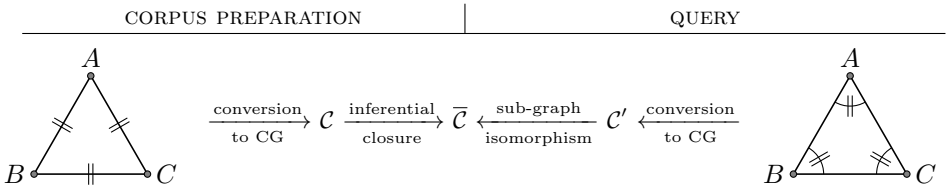

Fig. 2. Geometric Query Using Conceptual Graphs

For each construction in *TGTP* the conceptual graphs is found, then its inferential closure is calculated. When a query is done the conceptual graph is found, then a intermediate representation (see Section 2.4) is used as a filter to found a list of potential candidates and the (still to be implemented), using a sub-graph isomorphism the matches would be found.

2.2 Figures Represented as Conceptual Graphs

We use concepts point, segment, line, circle, angle, ratio. The model semantics of this vocabulary are the corresponding geometric notions (interpretations of the former four concepts take their values in $\mathbb{R}^2$, the interpretation of angle is a full-angle ([2, p. 44–50]) and interpretation of ratio is a real number). We also use three constants: angle:0 and angle:1 are angle-type individuals corresponding to full-angles $\angle[0]$ and $\angle[1]$ (in Chou notation), and ratio:1 is a ratio individual of value 1. The relations of our vocabulary are the following (see Table 1):

In conceptual graphs representing geometric figures, every segment has to be connected to a single line by an is_contained_in relation and every pair of lines ℓ, ℓ' is connected to angle concepts $\angle[\ell, \ell']$ and $\angle[\ell', \ell]$ by is_angle_of relations. These angle concepts are interconnected by the is_negative_of relation. When two lines are geometrically parallel, their angle is the individual angle:0; when two lines are geometrically perpendicular, their angle is the angle:1 individual. Every pair of segments is connected to a ratio by an is_ratio_of$_s$ relation; when they are geometrically congruent, then their ratio is the ratio:1 individual. If point:C is the geometric midpoint of segment:AB then three segments AB, AC

relation	arity	signature	relation	arity	signature
is_incident_to$_s$	2	(point,segment)	is_negative_of	2	(angle,angle)
is_incident_to$_c$	2	(point,circle)	is_ratio_of$_s$	3	(ratio,segment,segment)
is_contained_in	2	(segment,line)	is_ratio_of$_a$	3	(ratio,angle,angle)
is_center_of	2	(point,circle)	is_equal_to$_a$	2	(angle,angle)
is_angle_of	3	(angle,line,line)	is_equal_to$_r$	2	(ratio,ratio)
is_summit_of	2	(point,angle)			

Table 1. Geometric Relations

and BC have to be provided in the graph and the ratio of AC and BC is ratio:1. As for is_equal_to relations, they are only used between angles or between ratios. Consistency checking algorithms continuously verify that equality is transitive and that for every path containing an even number of is_negative_of relations there will an equality relation. Integrity checking algorithms verify that every segment is connected to a single line, that lines with zero angle and a common point are merged, that every circle has a single center point, etc.

2.3 Inferential Closure

We have implemented the inference rules as Python functions. They are then repeatedly applied until the CG remains unchanged, which means inferential closure has been attained. We have adapted rules specific to full-angles to this format, as well as rules D1–D75 of [3, p. 242].

The fact that we use CGs allows us to be independent of predicate argument order: for example, the sole purpose of rules D14–D17 in [3] is to ensure that the predicate cyclic(A, B, C, D) is true for any order of arguments A, B, C, D. In our case, we get four point concepts connected to the same circle concept via a is_incident_to relation, without any order. This allows us to significantly reduce the number of inference rules to apply, compared to [3].

Applying an inference rule is finding a pattern in the graph (i.e., a CG λ-rule, see [1, Ch. 10]) and transforming the graph in a specific way (by adding or merging vertices and/or edges). To avoid unnecessary use of the sub-graph isomorphism algorithm, the system calculates global trail distributions of inference rule patterns so that they are applied only if there is a chance that they will indeed match some sub-graph and transform it.

2.4 Global Trail Distributions

The inferential closure and the sub-graph isomorphism algorithms are heavy CPU consumers, we have developed a strategy for finding sets of potential matches, so that the set of figures to which the algorithms has to be applied is as small as possible.

To allow easier searching of match candidates in the corpus, a sequence of numeric values, called *global trail distribution*, is calculated out of the query graph.

A *global trail distribution*[5] is a sequence of key/value pairs partially describing the query conceptual graph. It has the important property that if the query graph is indeed contained in some corpus graph, then (a) *all keys of the query graph must also be keys of the corpus graph*, and (b) *the value of every key of the query must be smaller or equal to the value of the same key in the corpus graph*. Verification of graph compatibility is straightforward: we check whether all query keys also belong to the corpus graph key set, if not then the corpus graph is removed from the candidate list. Once this test is passed we compare values of query keys and corpus keys and check whether inequality is verified in all cases.

Keys are obtained in the following way: we take *all trails* of the CG. Let $p = (e_1, \ldots, e_n)$ be a trail (and hence $e_i \neq e_j$ for all $1 \leq i < j \leq n$), where e_i are concept nodes, relation nodes and relation edge labels. If we replace concepts by their types and relations by their symbols, we get a sequence $(t_1, \ldots, t_n)$ of concept types, relation symbols and relation edge labels. We replace t_is by numeric identifiers and call the obtained numeric sequence $\tau(p)$. Sequence $\tau(p)$ describes the specific trail p but also all other trails containing similar concepts and relations in the same order. The fact is that if we want the query graph to be contained in the corpus graph, then the latter must also contain at least the same number of trails of the same type sequence as the former. Let $\#\tau(p) = |\{p' \in \text{Trails}(G) : \tau(p) = \tau(p')\}|$ where $\text{Trails}(G)$ is the set of trails of G. We use $\tau(p)$ sequences as keys and $\#\tau(p)$ as values, i.e., the *global trail distribution* $\text{GTD}(G)$ *of graph* G is defined as:

$$\text{GTD}(G) = \{(\tau(p), \#\tau(p)) : p \in \text{Trails}(G)\}$$

3 Geometric Search Implementation in *TGTP*

The implementation of the geometric search in *TGTP* is divided in two steps with several sub-steps:

1. corpus preparation (to be done once for each figure in *TGTP*)[6]:
   (a) convert the corpus into conceptual graphs. This conversion is very efficient, $0.54s \leq t \leq 5.55s$, for the examples timed, with an average value of $1.34s$;
   (b) obtain the inferential closure of each figure in the corpus. This can be very heavy time consuming process, ranging from $0.91s$ to $> 100000s$. This step is still not completed, not all the figures have a corresponding conceptual graphs, inferential closure.
2. the query (to be done for every query):
   (a) use a DGS (*GeoGebra*, JavaScript applet) to make the query;
   (b) convert the query into a conceptual graph (very efficient, see 1a);

[5] In graph theory, a trail is a path with no repeated edges.

[6] It should be noted that some geometric conjectures can be in *TGTP* without a corresponding figure, For those cases the geometric search is not applicable

(c) compare the global trail distribution of the query with those in the corpus, obtaining a set of candidates. Using a *MySQL right outer join* query [6] this is done very quickly (few seconds);
(d) apply typed sub-graph isomorphism algorithm to candidates. This step is still to be implemented;
(e) if the algorithm succeeds, return a list of corpus graphs as standard geometry figures representations. Using the same DGS used for making the query, the *TGTP*'s user should be able to browse the list of results. For each geometric construction in the list it will be possible to visualise it, with the part matching the query highlighted.

In *TGTP* the queries are constructed using *GeoGebra*, the global trail distributions for that construction are calculated and then matched against the ones in the corpus. This provides a very fast mechanism to build a list of similar constructions that is made available to the *TGTP* user making the query.

4 Future Work and Conclusion

The *TGTP* system has already fulfilled many of the goals specified at the beginning of the project. The geometric search mechanism is, in our opinion, a very interesting addition to the platform. Nevertheless there are still many improvements to be done.

For now all the searches are independent of each others, the user should be able to combine them, e.g. after a given full-text search, run a geometric search in the resulting list.

Stopping the query at the global trail distribution matching level, is a very fast way to get results, but the price to pay is uncertainty about precise matching and impossibility to highlight the query match inside the corpus graph. The next steps, still to be implemented in *TGTP*, are: to improve the filtering of the result list (step 2c) with a deep learning generic graph representation learning framework [8]; after having the result list of problems we should be able to apply the typed sub-graph isomorphism, the result could then be visualised using *GeoGebra*, with colours marking the query construction as a sub-construction of the corpus constructions.

References

1. Chein, M., Mugnier, M.L.: Graph-based Knowledge Representation. Computational Foundations of Conceptual Graphs. Advanced Information and Knowledge Processing Series, Springer (2009)
2. Chou, S.C., Gao, X.S., Zhang, J.Z.: Machine Proofs in Geometry. World Scientific (1994)
3. Chou, S.C., Gao, X.S., Zhang, J.Z.: A deductive database approach to automated geometry theorem proving and discovering. Journal of Automated Reasoning 25, 219–246 (2000)

4. Haralambous, Y., Quaresma, P.: Querying geometric figures using a controlled language, ontological graphs and dependency lattices. In: et al., S.W. (ed.) CICM 2014. LNAI, vol. 8543, pp. 298–311. Springer (2014)
5. Mathis, P., Thierry, S.E.B.: A formalization of geometric constraint systems and their decomposition. Formal Aspects of Computing 22(2), 129–151 (Mar 2010), https://doi.org/10.1007/s00165-009-0117-8
6. Oracle: MySQL 5.5 Reference Manual. Oracle, 5.5 edn. (January 2011), revision: 24956
7. Quaresma, P.: Thousands of Geometric problems for geometric Theorem Provers (TGTP). In: Schreck, P., Narboux, J., Richter-Gebert, J. (eds.) Automated Deduction in Geometry, Lecture Notes in Computer Science, vol. 6877, pp. 169–181. Springer (2011)
8. Rossi, R.A., Zhou, R., Ahmed, N.K.: Deep inductive network representation learning. In: Companion Proceedings of the The Web Conference 2018. pp. 953–960. WWW '18, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland (2018), https://doi.org/10.1145/3184558.3191524