2015

Synthesis With Hypergraphs

Christopher Thomas Alvin
Louisiana State University and Agricultural and Mechanical College

SYNTHESIS WITH HYPERGRAPHS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Computer Science

by
Christopher T. Alvin
B.A., Ripon College, 1999
M.S., University of Wisconsin at Madison, 2001
M.S., Marquette University, 2011
August 2015

Acknowledgments

I would like to thank Supratik Mukhopadhyay for his efforts and creativity as my advisor. I would also like to thank Louisiana State University and the Economic Development Assistantship. To Sumit Gulwani for his creative ideas in intelligent tutoring. I am indebted to Rupak Majumdar for a chance walk in the Alps, advice, and tremendous ability to simplify complex notions. I also need to thank Michal Brylinski and Jimmie Lawson for their ideas as well as the Misagh Naderi and Brian Peterson for their collaborations.

Finally, I must thank Lori for putting up with me; it is not an easy set of challenges.

Table of Contents

Abstract

Many problems related to synthesis with intelligent tutoring may be phrased as program synthesis problems using AI-style search and formal reasoning techniques. The first two results in this dissertation focus on problem synthesis as an aspect of intelligent tutoring systems applied to STEM-based education frameworks, specifically high school geometry. Given a geometric figure as input, our technique constructs a hypergraph representing logical deduction of facts, and then traverses the hypergraph to synthesize problems and their corresponding solutions.

Using similar techniques, our third result is focused on exhaustive synthesis of molecules. This synthesis process involves bonding sets of basic, molecular 'fragments' according to chemical constraints to create molecules of increasing size. For each input set of fragments, synthesis results in a significant set of molecules. Due to big data constraints we give special consideration in how to construct a corresponding molecular hypergraph based on a target, template molecule. Synthesis of the target molecule in a laboratory environment then corresponds to any path in the molecular hypergraph from the set of fragments to the target molecule.

Chapter 1
Introduction

Program synthesis is the task of automatically discovering an executable piece of code when given constraints through demonstrations, input-output pairs, or other example-based input. Many problems may be phrased as program synthesis problems using AI-style search and formal reasoning techniques; in this dissertation we focus on two distinct synthesis problems. Specifically, we will focus on the construction and exploration of hypergraphs for problem and solution synthesis in intelligent tutoring systems as well as synthesis of molecular compounds. In Chapter 3 and Chapter 4 we describe problem and solution synthesis as applied to STEM-based education frameworks, specifically high-school geometry. In Chapter 5, we apply some similar techniques to the space of molecules with the goal of providing the theoretical foundation and toolset for discovery of new antibiotic / antimicrobial compounds.

1.1 Geometry Problem and Solution Synthesis

With the advent of visualization technologies (tablets, graphing calculators, etc.) in the classroom, there has been a shift in mathematics teaching where a problem is viewed from multiple perspectives: graphical, numerical, and algebraic. High school geometry is particularly interesting in this regard because it combines the implicit visual perspective and deductive logic skills. Many online teaching and learning tools exist for high school mathematics courses through Calculus; however there is a limit to the number and types of problems a student may use for practice or a teacher may use for test generation.

On-demand generation of new problems that have specific problem and solution characteristics (such as difficulty level, use of a certain set of concepts, etc.) is a difficult task for any teacher. The ultimate goal for problem synthesis is effective student learning, but automating problem synthesis has several benefits including efficient construction of homework and exams, facilitating effective differentiated instruction, and avoiding copyright issues encountered with textbooks or other copyrighted materials.

In Chapter 3 we present a semi-automated methodology and tool, *GeoTutor* [7], for generating geometric proof problems of the kind found in a high-school curriculum. We formalize the notion of a geometry proof problem and describe an algorithm for generating such problems over a user-provided figure. Our experimental results indicate that our problem generation algorithm can effectively generate proof problems in elementary geometry. On a corpus of 110 figures taken from popular geometry textbooks, our system generated an average of about 443 problems per figure in an average time of 4.7 seconds per figure.

In Chapter 4, we present a tool, *GeoShader* [8], that not only solves *shaded area* geometry problems but also synthesizes such problems. We consider three distinct use cases:

1. given a geometric figure and a shaded region within it, solve the problem by calculating the area of the shaded region,
2. given a geometric figure, synthesize all possible *interesting* shaded area problems from it, and
3. given a set of shapes (e.g., triangles, circles, etc.) compose them in all possible configurations (e.g., one shape inside another, one shape adjoining another, etc.) to synthesize a geometric figure that provides interesting shaded area problems.

On a corpus of 102 problems taken from popular geometry textbooks, *GeoShader* successfully solved the original problem and generated an average of 257 problems per figure in an average time of 13.4 seconds per figure. Given a set of three polygons, we synthesized 3533 figures resulting in a mean of 16.5 interesting problems per figure.

1.2 Molecular Synthesis

According to the Centers for Disease Control (CDC), antibiotic / antimicrobial resistance is a significant threat that results in at least 23,000 deaths each year [37]. In order to combat the worldwide epidemic of antimicrobial resistance, we describe a *molecular synthesis* process that, given a set of basic molecular building blocks (molecular fragments), we perform an exhaustive synthesis in order to construct all possible molecules using those

constituent molecular fragments. We then discuss validation of our synthesis techniques and give evidence that our implementation tool *Synth* is accurate, efficient, and can explore deep in the chemical compound search space in a short amount of time thus facilitating discovery of new drug compounds.

Chapter 2
Hypergraphs

In each synthesis space we must decide how to encode that information in a target data structure. The synthesis efforts we describe in this thesis generally require deduction of facts. For example, in order to represent logical deduction in a directed graph data structure, we must define correspondences among nodes and edges as well as operations such as paths and reachability. In this section, we first consider a directed graph data structure before expanding into a hypergraph [13], a generalizeation of a graph data structure.

2.1 Graphs

We may define a *directed graph* based on sets of nodes and edges connecting nodes.

Definition 1 (Directed Graph)**.** *A* directed graph $G(N, E)$ *is a data structure where* N *is a set of nodes and* E *a set of directed edges. Each directed edge* $e \in E$ *is defined by the ordered pair* $e = (s, t)$ *where* $s, t \in N$*; we may refer to* s *as the* source node *and* t *as the* target node.

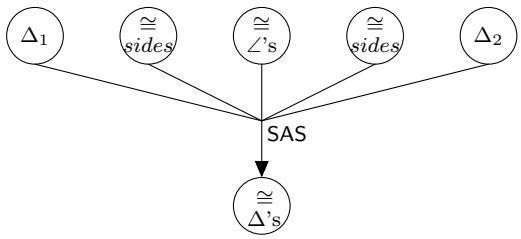


Figure 2.1: Logical Deduction of Triangle Congruence using SAS

However, for a directed graph $G(N, E)$ where, for all $n \in N$, n corresponds to a singleton fact, the graph data structure is limiting. General deduction of a single fact often arises from many antecedent facts. For example, as shown in Figure 2.1, the Side-Angle-Side (SAS) geometry congruence axiom requires three facts relating two triangles in order to deduce the single fact that the two triangles are congruent. Our synthesis efforts hence require a more general, many-to-one relationship among facts; thus we require a hypergraph structure.

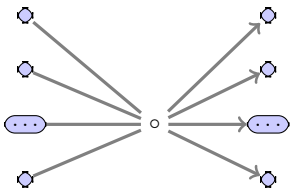


Figure 2.2: A Many-To-Many Directed Hyperedge.

2.2 Synthesis Hypergraph

We first consider a many-to-many hypergraph structure in Definition 2 before focusing on a special case of a hypergraph we use in our synthesis efforts in Definition 3.

Definition 2 (General, Directed Hypergraph)**.** *A* directed hypergraph (N, E) *is a data structure where* N *is a set of hypernodes and* E *a set of directed hyperedges. Each directed hyperedge* $e \in E$ *is defined by the ordered pair* $e = (S, T)$ *where* $S, T \subseteq N$*.*

In a deductive domain, it is not necessary to adopt a many-to-many directed hyperedge as defined in Definition 2 and shown in Figure 2.2. We instead use a hypergraph in which hyperedges consist of many source nodes and a single target node as shown in Figure 2.1 and defined in Definition 3.

Definition 3 (Synthesis Hypergraph)**.** *A* synthesis hypergraph *is a directed hypergraph* $H\,(N, E_{\mathcal{A}})$ *where* N *is a set of hypernodes and* E *a set of directed hyperedges over a set of annotations* $\mathcal{A}$*. Each directed hyperedge* $e \in E$ *is defined by the ordered pair* $e = (S, t)$ *where* $S \subseteq N$ *and* $t \in N$*.*

2.3 Hyperpaths and Hyper-Reachability

In each of our synthesis efforts, we seek correspondence with the nodes and hyperedges of our synthesis hypergraph as well as operations on those hypergraphs. Specifically, we are most interested in hyperpaths and hyper-reachability. For completeness purposes, we define these concepts with respect to a synthesis hypergraph in Definition 6 and Definition 7, but first we define the necessary structures to acquire hyperpaths.

To simplify this path-finding process we define a 'reverse' structure of a synthesis hypergraph by first defining a transpose hyperedge (Definition 4) and secondly the dual of a synthesis hypergraph (Definition 5). We note that the dual of a synthesis hypergraph is a directed graph as stated in Definition 1 where operations on graphs such as path and reachability are well-defined [26].

Definition 4 (Transpose Hyperedge)**.** *For a hyperedge $e = (S, t)$ with source nodes S and target node t, the corresponding* transpose hyperedge *is a set of edges given by $e^T = \{(t, s) : \forall s \in S\}$.*

Definition 5 (Synthesis Hypergraph Dual)**.** *Let $H\,(N, E_{\mathcal{A}})$ be a synthesis hypergraph with nodes N and hyperedges E over a set of annotations $\mathcal{A}$. The* dual of an analysis hypergraph*, $H^T(N, \mathcal{E})$, is a graph with nodes N and edges defined by $\mathcal{E} = \bigcup_{e \in E} e^T$, the union of all transpose hyperedges of E.*

In simple terms, the dual of a synthesis hypergraph is a graph with the same set of nodes and the hyperedges of the hypergraph are split into one-to-one edges with the directions reversed. We may now easily define a hyperpath in a synthesis hypergraph using the dual of a synthesis hypergraph.

Definition 6 (Hyperpath)**.** *Let $H\,(N, E_{\mathcal{A}})$ be a synthesis hypergraph, $I \subset N$, and $g \in N$. The* hyperpath *from I to g is the set of hypernodes and hyperedges corresponding to the nodes and edges of the path from g to each $f \in I$ in H^T. We say that the* shortest hyperpath *from I to g is the hyperpath that uses the fewest number of hyperedges.*

We can now easily define hyper-reachability in a synthesis hypergraph.

Definition 7 (Hyper-Reachability)**.** *Let $H\,(N, E_{\mathcal{A}})$ be a synthesis hypergraph, $I \subset N$, and $g \in N$. Then g is* hyper-reachable *from I if there exists a hyperpath from I to g.*

Both of these operations will play a critical role in each of our forthcoming results in Chapter 3 through Chapter 5.

2.4 Sub-Hypergraph Selection through Pebbling

In a synthesis hypergraph $H\,(N, E_{\mathcal{A}})$ each hyperedge is annotated with a parameterized set of values $A \in \mathcal{A}$ defined in the synthesis space. Edge annotations provides the user with an ability to exclude a set of hyperedges and restrict the corresponding set of syntheses in the synthesis space. We more formally define this notion in a pebbled synthesis hypergraph in Definition 8 as computed using Algorithm 2.1, a process we informally call *pebbling*.

Algorithm 2.1 Sub-Hypergraph Selection Through Pebbling

```
1:  procedure PEBBLE(Hypergraph H (N, E_A), N_P ⊆ N, A_P ⊆ A)
2:      Hypergraph P                                   ▷ Pebbled Sub-Hypergraph
3:      Worklist W ← N_P
4:      while !W.empty() do
5:          n ← W.dequeue()                            ▷ Acquire a node
6:          if !n.pebbled() then
7:              n.pebble()                             ▷ Mark the node
8:              P.AddNode(n)
9:              for all e ∈ n.edges do
10:                 ▷ Consider only allowable hyperedges
11:                 if e.annotation ∈ A_P then
12:                     ▷ If all hyperedge source nodes are pebbled, add target to worklist
13:                     ▷ to propogate forward
14:                     if e.pebbled() then
15:                         P.AddHyperedge(e)
16:                         W.enqueue(e.target)
17:                     end if
18:                 end if
19:             end for
20:         end if
21:     end while
22:     return P
23: end procedure
```

Definition 8 (Pebbled Synthesis Hypergraph)**.** *Let $H\,(N, E_{\mathcal{A}})$ be a synthesis hypergraph with $N_P \subseteq N$ a subset of nodes and a subset of annotations $\mathcal{A}_P \subseteq \mathcal{A}$. Then $H_P\,(N_P, E_{\mathcal{A}_P})$ is a* pebbled synthesis hypergraph *containing only reachable nodes and hyperedges as dictated by N_P and $\mathcal{A}_P$, respectively.*

Algorithm 2.1 is a modification of the classic algorithm marking algorithm as first defined by Dowling and Gallier [32] for satisfiability of propositional horn clauses. Pebbling is a linear-time traversal over a synthesis hypergraph that identifies the sub-hypergraph [13] that satisfies the constraints stated by the user. As described in Algorithm 2.1, pebbling is a breadth-first traversal over a synthesis hypergraph where we mark each node with a pebble once it is visited (Line 7). Then on Line 9 through Line 19 we use the following rule for pebbling and propogation: if all source nodes of a hyperedge are pebbled, we place the target node of the hyperedge in the work list. As pebbling continues, we add all pebbled nodes (Line 8) and hyperedges (Line 15) to the sub-hypergraph in preparation for the return of the pebbled version of the hypergraph (Line 22).

Chapter 3
Synthesis of Geometry Proof Problems and Their Solutions

This chapter presents a semi-automated methodology for generating geometric proof problems of the kind found in a high-school curriculum. We formalize the notion of a geometry proof problem and describe an algorithm for generating such problems over a user-provided figure. Our experimental results indicate that our problem generation algorithm can effectively generate proof problems in elementary geometry. On a corpus of 110 figures taken from popular geometry textbooks, our system generated an average of about 443 problems per figure in an average time of 4.7 seconds per figure.

3.1 Introduction

Learning in mathematics is more deeply rooted when a student is able to view a problem from multiple perspectives: graphically, numerically, and algebraically. High school geometry is particularly interesting in this regard because it combines the implicit visual perspective and deductive logic skills. For some, geometry is the favorite mathematics course in high school because of the combination of the implicit visual perspective and the constant exercising of deductive logic skills. This chapter presents a technology to enhance geometry education. In particular, we present a technique for automatically generating fresh geometry proof problems from the figures of given problems.

Generating fresh problems that have specific solution characteristics (such as difficulty level, use of a certain set of concepts) is a difficult task for educators. Automating problem generation has several benefits. First, it can help avoid copyright issues. It is illegal to make photocopies of a textbook and may not be legal to publish an original problem from a textbook on a course website. A problem generation tool can provide instructors with fresh problems (that have characteristics similar to that of the original problem) for use in their assignments, exams, or lecture notes. Second, it can help prevent cheating in classrooms or online education platforms (with unsynchronized instruction) since each student can be provided with a different problem but with the same characteristics. Third, it can be used

to generate personalized workflows for students. If a student solves a problem correctly, then the student may be presented with a problem that is more difficult than the last problem, or exercises a richer set of concepts. If a student fails to solve a problem, then the student may be presented with simpler problems to identify, reinforce, and master core concepts.

We formalize the notion of a geometry proof problem, which consists of a figure, some assumptions about the figure, goals that need to be established about the figure, and the set of axioms that need to be used. We propose a semi-automated methodology for generating such problems. Given a figure and a set of axioms, our problem generation technique produces a set of problems over that figure in the form of pairs of assumptions and goals. Such problems, generated across a large set of figures provided by the user, can be stored in a database along with their characteristics. This empowers users to query the database with specific characteristics to obtain custom problems.

Our problem generation technique operates in three steps. First, it produces a logical geometry hypergraph (Definition 14) that represents all possible proofs for all possible problems over a given pair of user-provided figures and axioms. The hypergraph construction requires enumerating all facts that are true of the figure as nodes in the hypergraph. Furthermore, a set of source facts is connected to a target fact using a directed hyperedge labeled with a user-provided axiom if the axiom can be used to deduce the target fact from the source facts. Then, the tool systematically enumerates all minimal sets of assumptions (Algorithm 3.1). An assumption is a fact about the figure, and informally, a set of assumptions is minimal if every assumption is non-redundant. Finally, for any minimal set of assumptions I, the tool systematically enumerates all possible goal sets G such that (I, G) is an interesting problem (Algorithm 3.2).

We evaluated the effectiveness of our problem generation algorithm on 110 figures taken from various geometry textbooks. Our algorithm generated an average of 443 problems in

an average time of 4.7 seconds per figure. We also observed that there were several problems with same characteristics across various figures.

This chapter makes the following contributions:

- We informally describe the geometry proof problem synthesis domain (§3.2).
- We then formalize a geometric figure as a partial ordering of geometric classes as well as the notion of a geometry problem (§3.3).
- We then motivate problem generation interfaces corresponding to characteristics of geometry proof problems (§3.5).
- We present a technique for generating proof problems over a given geometric figure (§3.4).
- We describe experimental results illustrating the efficacy of our problem generation interfaces and our problem generation algorithm (§3.6).

3.2 Informal Theoretical Foundations in Euclidean Geometry

Informally, a geometric figure is a pictorial representation of a collection of geometric objects (points, lines, circles) in a specific orientation with each other. Internally, we represent geometric figures using first-order logic constraints which can be derived by analyzing a pictorial representation. We work in a first order language with arithmetic and constants ranging over points. We omit a full description of the logical language and illustrate it through examples. Our logic consists of relations such as betweenness $\mathsf{Between}(A, B, C)$ (which implies collinearity of points A, B, and C), congruence, and equality relations on line segments or angles. For ease of readability, in the following examples, we also use derived predicates such as $\mathsf{Triangle}(A, B, C)$ (the three points form a triangle, denoted ΔABC), $\mathsf{Collinear}(A, B, C)$ (points are collinear), $\mathsf{RightAngle}(A, B, C)$, etc.

We compute internal representations from pictorial representations of a figure. We assume that input figures are drawn to scale but the problem instances we generate will not assume that figures are drawn to scale. Thus, in the internal representation for a

Table 3.1: Example Set of Geometric Axioms

Axiom Name	**Premise(s)**	**Conclusion(s)**
Midpoint Definition	Midpoint(M, $\overline{AB}$)	$AM = MB$
Angle Addition	$\angle ABC, \angle CBD$ Exterior($D, \angle ABC$)	$\angle ABC + \angle CBD = \angle ABD$
Vertical Angles	Intersect($X, \overline{AB}, \overline{CD}$)	$\angle AXD \cong \angle CXB$, $\angle AXC \cong \angle BXD$
Side-Side-Side	$\Delta ABC, \Delta DEF$, $\overline{AB} \cong \overline{DE}, \overline{BC} \cong \overline{EF}$ $\overline{CA} \cong \overline{FD}$	$\Delta ABC \cong \Delta DEF$
Alternate Interior Angles	$\overline{CD} \parallel \overline{EF}$, Intersect($M, \overline{AB}, \overline{CD}$), Intersect($N, \overline{AB}, \overline{EF}$)	$\angle ENM \cong \angle NMD$, $\angle FNM \cong \angle NMC$

figure Fig, we distinguish between *implicit* and *explicit* facts. Implicit predicates only provide orientation (or "betweenness") information but not relationships on measurements. Explicit predicates provide relations based on measurement and may not hold when the figure is distorted. For example, implicit predicates would state that ABC is a triangle or that line segments AB and CD intersect at M, and explicit predicates would state $AB = CD$ or $\angle ABC$ is a right angle. Technically, implicit predicates are those facts about the figure provable in *ordered geometry* [27], and explicit predicates are those facts provable in Euclidean geometry minus the implicit ones. For a figure Fig, we write $\mathcal{I}(\mathsf{Fig})$ for the set of implicit facts and $\mathcal{E}(\mathsf{Fig})$ the set of explicit facts.

We may now formalize the definition of a geometry axiom as a mechanism that uses a set of facts to derive a new target fact.

Definition 9 (Geometry Axiom)**.** *A* geometry axiom *is a Horn clause whose ground instances are implicit or explicit predicates and consists of a set of* premises *and a* conclusion.

The free variables in a geometry axiom are (implicitly) universally quantified. Given an axiom A, we say that A *derives* a predicate p from a set P of predicates if there is an instantiation of the premises of A with P and the conclusion with p: $P \vdash^A p$. Table 3.1 gives some examples of geometry axioms.

Definition 10 (Geometry Problem)**.** *Let* Fig *be a figure,* $\mathcal{I}(\mathsf{Fig})$ *be the set of implicit facts, and* Axm *be a set of geometry axioms. A* geometry (proof) problem *over* $(\mathsf{Fig}, \mathsf{Axm})$ *is a pair* (I, G)*, where the assumptions* $I \subseteq \mathcal{E}(\mathsf{Fig})$ *and goals* $G \subseteq \mathcal{E}(\mathsf{Fig})$ *are sets of explicit facts such that* $I \cap G = \emptyset$ *and* $\mathcal{I}(\mathsf{Fig}) \cup I \cup \mathsf{Axm}$ *imply each* $g \in G$ *using first-order reasoning.*

In Definition 10 we require the disjointness condition between I and G to ensure problems are non-trivial, and the derivation condition to ensure problems have solutions. Given a geometry problem, we may now define a converse geometry problem.

Definition 11 (Converse Geometry Problem)**.** *The* converse *of a problem* (I, G) *over* $(\mathsf{Fig}, \mathsf{Axm})$ *is the problem* (G, I) *over* $(\mathsf{Fig}, \mathsf{Axm})$*, if it is indeed a problem.*

We note that a corresponding converse problem may not exist for a given (I, G) over $(\mathsf{Fig}, \mathsf{Axm})$. We may now define concepts related to the quality of a geometry problem.

Definition 12 (Strict, Interesting Geometry Problem)**.** *A geometry problem* (I, G) *over* $(\mathsf{Fig}, \mathsf{Axm})$ *is* interesting *if no strict subset of* I *together with* $\mathcal{I}(\mathsf{Fig})$ *can establish every goal in* G *using* Axm*. An interesting problem is* strict *if* G *is minimal, i.e.,* (I, G') *is not interesting for any strict subset* $G' \subsetneq G$*.*

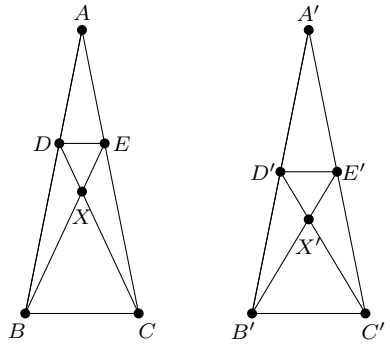
Observe that an interesting problem where G is a singleton is strict.

Definition 13 (Complete Geometry Problem)**.** *An interesting geometry problem* (I, G) *over* $(\mathsf{Fig}, \mathsf{Axm})$ *is* complete *if for any predicate* $p \in \mathcal{E}(\mathsf{Fig})$*,* $\mathcal{I}(\mathsf{Fig}) \cup I \cup \mathsf{Axm}$ *derives* p*.*

A complete problem is strict if it is not complete for any strict subset G' of G. Figure 3.1 gives some examples of interesting and complete geometry problems.

Let (I, G) be a problem over $(\mathsf{Fig}, \mathsf{Axm})$. A proof that $\mathcal{I}(\mathsf{Fig}) \cup \mathsf{Axm} \cup I$ derives G consists of first-order derivations, one for each $g \in G$, whose root is labeled g, whose leaves are elements of $\mathcal{I}(\mathsf{Fig}) \cup \mathcal{E}(\mathsf{Fig})$ and whose internal nodes are obtained by instantiating

Let Axm be a common set of geometric axioms in figures $(\mathsf{Fig}_1, \mathsf{Axm})$ and $(\mathsf{Fig}_2, \mathsf{Axm})$.



$(\mathsf{Fig}_1, \mathsf{Axm})$ $(\mathsf{Fig}_2, \mathsf{Axm})$

For both figures the original textbook problem is stated as (I, G) where

$$I = \{\triangle ABE \cong \triangle ACD\}$$

and

$$G = \{\triangle ADE \sim \triangle ABC\}.$$

Fig_1 is indistinguishable from Fig_2 except points D, D', E, E', and consequently X, X'. Specifically, in Fig_2 D' is the midpoint of segment $\overline{A'B'}$; similarly E' is the midpoint of $\overline{A'C'}$. That is, $\mathcal{I}(\mathsf{Fig}_1) = \mathcal{I}(\mathsf{Fig}_2)$ while $\mathcal{E}(\mathsf{Fig}_1) \neq \mathcal{E}(\mathsf{Fig}_2)$ since $\left\{\text{Midpoint}(D', \overline{A'B'}), \text{Midpoint}(E', \overline{A'C'})\right\} \subset \mathcal{E}(\mathsf{Fig}_2)$.

For $(\mathsf{Fig}_1, \mathsf{Axm})$ and $(\mathsf{Fig}_2, \mathsf{Axm})$ we will generate the exact same set of problems $(I, \{g_1\})$ where $I = \{\Delta ABE \cong \Delta ACD\}$ and g_1 may be any of the following propositions.

- $\triangle ADE \sim \triangle ABC$
- $\triangle BCX$ is Isosceles
- $\angle DEA \cong \angle CBA$
- $\angle BCD \cong \angle CBE$
- $\triangle DEX$ is Isosceles
- $\triangle BDX \cong \triangle CEX$
- $\triangle DBC \cong \triangle ECB$
- $\overline{DE} \parallel \overline{BC}$

I completely defines Fig_1; hence all problems $(I, \{g_1\})$ are *strictly complete problems.* I does not define Fig_2 since it is not possible to prove $\text{Midpoint}(D', \overline{A'B'})$ nor $\text{Midpoint}(E', \overline{A'C'})$. For Fig_2, all problems in $(I, \{g_1\})$ are simply *interesting problems.*

Figure 3.1: Example of Strictly Interesting and Strictly Complete Problems

an axiom from Axm. Our problem generation algorithm will search through many proofs. Hence, we use a hypergraph representation for all possible derivations. Since the set $\mathcal{I}(\mathsf{Fig})$ is fixed, we do not represent nodes for them.

Definition 14 (Logical Geometry Hypergraph)**.** *For a pair* $(\mathsf{Fig}, \mathsf{Axm})$*, the* logical geometry hypergraph $H(\mathsf{Fig}, \mathsf{Axm})$ *is a synthesis hypergraph whose nodes consist of all predicates in* $\mathcal{E}(\mathsf{Fig})$ *and whose edges are of the form* (P, p, A)*, where* $P \subseteq \mathcal{E}(\mathsf{Fig})$ *is a set of explicit predicates,* $p \in \mathcal{E}(\mathsf{Fig})$ *is an explicit predicate, and* $A \in \mathsf{Axm}$*, such that there exists a set* $Q \subseteq \mathcal{I}(\mathsf{Fig})$ *such that* A *derives* p *from* $P \cup Q$*.*

We then say reachability in the logical geometry hypergraph corresponds to logical derivability. For a set $T \subseteq \mathcal{E}(\mathsf{Fig})$, we define

$$Derive(T) = \{g \in \mathcal{E}(\mathsf{Fig}) \mid T \cup \mathcal{I}(\mathsf{Fig}) \cup \mathsf{Axm} \models g\}.$$

The set $Derive(T)$ coincides with the set of nodes reachable in the hypergraph $H(\mathsf{Fig}, \mathsf{Axm})$ starting from the set T of nodes. Thus, $Derive(T)$ can be computed for every set $T \subseteq \mathcal{E}(\mathsf{Fig})$ in time polynomial in the size of the hypergraph.

3.3 Formal Theoretical Foundations in Euclidean Geometry

In this section we consider a more formal discussion of the framework for problem synthesis in Euclidean geometry. In these discussion, we assume immutable figures in which the properties of that figure are not allowed to be modified nor any new information constructed.

3.3.1 Geometric Classes

There are several distinct types of objects in Euclidean geometry, most notably: points, rays, segments, lines, triangles, quadrilaterals, and circles. For our purposes, we define a class for each geometric object: the class of points $\mathcal{P}$, the class of segments $\mathcal{S}$, the class of triangles $\mathcal{T}$, etc.

Since points are considered to be the framework for which Euclidean geometry is founded, the only characteristic we will impose on a point is a coordinate in n dimensions ($n \geq 2$); that is, we coordinatize our geometry even if it is not apparent to the user. This also implies coordinate axes for the user interface even though they may be transparent to the user. As our focus is high school Euclidean geometry, we will restrict our

notion of a point to two or three dimensions as needed. We define the class of segments in terms of the class of points. We define the class of all triangles $\mathcal{T}$ as a collection of sets of three segments with the constraint that their intersections result in three unique points: the vertices of a triangle.

3.3.2 Theories and Figures

Let $\mathcal{L}$ be a logic [22] in which properties of a geometric figure are described. We assume a finite set of geometric classes including point, segment, triangle, isosceles triangle, and equilateral triangle. Let $\mathsf{Fig} = \{\mathsf{Fig}_1, \ldots, \mathsf{Fig}_k\}$ be the collection of k geometric classes. Also let Fig be a figure that belongs to a class Fig: formally, $\mathsf{Fig} \in \mathsf{Fig}$. We then define the theory of a class of figures Fig as $Th\,(\mathsf{Fig}) = \{\phi_1, \ldots, \phi_j\}$ where each ϕ_i is a property (a formula in $\mathcal{L}$) and $1 \leq i \leq j$ enumerate the minimal set of the implicit properties of Fig, $\mathcal{I}(\mathsf{Fig})$; i.e., $\forall \phi_i \in Th\,(\mathsf{Fig})\,, \{Th\,(\mathsf{Axm}) \cup Th\,(\mathsf{Fig}) \setminus \phi_i\} \nvDash \phi_i$ where Axm is the set of Euclid's axioms [51]. That is, $Th\,(\mathsf{Fig})$ consists of all properties of a class Fig that are innate to the class, but cannot be proven; in other words, implicit properties are those provable in ordered geometry [27]. For example, in the triangle class, one can neither prove that triangles have three segments nor prove that they have three internal angles. These are the implicit properties of the triangle class.

Ordering on Geometric Classes. The geometric classes defined in §3.3.1 give rise to an ordering among particular sets of classes. We first define the ordering operator and then prove that it implies a partial order on the set of geometric classes.

Definition 15 (Class Ordering Operator $\sqsubseteq$)**.** *We define the ordering operator $\sqsubseteq$ on classes as $\mathsf{Fig}_1 \sqsubseteq \mathsf{Fig}_2$ if and only if $Th\,(\mathsf{Fig}_1) \vDash Th\,(\mathsf{Fig}_2)$, i.e., if $Th\,(\mathsf{Fig}_1)$ logically entails $Th\,(\mathsf{Fig}_2)$.*

Proposition 1 (Partial Order of $\sqsubseteq$)**.** *$\sqsubseteq$ defines a partial order on Fig.*

Proof. Let $\mathsf{Fig}_c \in \mathsf{Fig}$. Then it is clear $Th\,(\mathsf{Fig}_c) \vDash Th\,(\mathsf{Fig}_c)$. Hence, $\sqsubseteq$ is reflexive. Let $\mathsf{Fig}_1, \mathsf{Fig}_2 \in \mathsf{Fig}$ with $\mathsf{Fig}_1 \sqsubseteq \mathsf{Fig}_2$ and $\mathsf{Fig}_2 \sqsubseteq \mathsf{Fig}_1$. It follows $Th\,(\mathsf{Fig}_1) \vDash Th\,(\mathsf{Fig}_2)$ and

$Th(\mathsf{Fig}_2) \vDash Th(\mathsf{Fig}_1)$. This implies that for the logical formulae $p_1, \ldots, p_k \in Th(\mathsf{Fig}_1)$ and $q_1, \ldots, q_\ell \in Th(\mathsf{Fig}_2)$, $\forall q_i, \exists\{p_j\} \vDash q_i$ and similarly $\forall p_i, \exists\{q_j\} \vDash p_i$. This implies $\mathsf{Fig}_1 = \mathsf{Fig}_2$ and thus $\sqsubseteq$ is antisymmetric.

Let $\mathsf{Fig}_1, \mathsf{Fig}_2, \mathsf{Fig}_3 \in \mathsf{Fig}$ with $\mathsf{Fig}_1 \sqsubseteq \mathsf{Fig}_2$ and $\mathsf{Fig}_2 \sqsubseteq \mathsf{Fig}_3$. By definition, $Th(\mathsf{Fig}_1) \vDash Th(\mathsf{Fig}_2)$ and $Th(\mathsf{Fig}_2) \vDash Th(\mathsf{Fig}_3)$. Since theories are logical formulae, it follows that Fig_1 is the set of logical formulae such that $Th(\mathsf{Fig}_1) \vDash Th(\mathsf{Fig}_3)$. Hence, $\mathsf{Fig}_1 \sqsubseteq \mathsf{Fig}_3$ and $\sqsubseteq$ is transitive. □

For a figure Fig to be described by a particular class Fig we say that the figure *forces* the theory of the class Fig: $\mathsf{Fig} \Vdash Th(\mathsf{Fig})$. Thus $\mathsf{Fig} \in \mathsf{Fig}$ if and only if $\mathsf{Fig} \Vdash Th(\mathsf{Fig})$. We now need to show that a figure cannot be an element in two distinct chains in the partial order; e.g. a figure cannot be both a triangle and circle.

Lemma 3.3.1 (Unique Figure Chain)**.** *For a figure* Fig *and classes* Fig_1 *and* Fig_2*, if* $\mathsf{Fig} \in \mathsf{Fig}_1$ *and* $\mathsf{Fig} \in \mathsf{Fig}_2$*, then either* $\mathsf{Fig}_1 \sqsubseteq \mathsf{Fig}_2$ *or* $\mathsf{Fig}_2 \sqsubseteq \mathsf{Fig}_1$*.*

Proof. Suppose without loss of generality $\mathsf{Fig}_1 \not\sqsubseteq \mathsf{Fig}_2$. By definition, $\mathsf{Fig} \Vdash Th(\mathsf{Fig}_1)$ and $\mathsf{Fig} \Vdash Th(\mathsf{Fig}_2)$. As $\mathsf{Fig}_1 \not\sqsubseteq \mathsf{Fig}_2$, there exists a logical formula $p \in \mathsf{Fig}_2$ such that $\mathsf{Fig} \Vdash Th(\mathsf{Fig}_1) \nvDash p$. As $\mathsf{Fig} \Vdash Th(\mathsf{Fig}_2) \vDash p$, this is a contradiction so $\mathsf{Fig}_1 \sqsubseteq \mathsf{Fig}_2$ as desired. □

We also require a figure to be defined by the most appropriate class.

Corollary 3.3.2. *For a figure* Fig*, there exists classes* Fig_b *and* Fig_B *such that for* $\mathsf{Fig} \in \mathsf{Fig}_b, \mathsf{Fig} \in \mathsf{Fig}_B$ *and for all* Fig' *such that* $\mathsf{Fig} \in \mathsf{Fig}'$*,* $\mathsf{Fig}_b \sqsubseteq \mathsf{Fig}' \sqsubseteq \mathsf{Fig}_B$*.*

Fig_b defines the greatest lower bound of classes for a figure Fig. We call Fig_b the *strongest* class corresponding to figure Fig and write $strong(\mathsf{Fig})$. Fig_B defines the least upper bound of classes for a figure Fig. We call Fig_B the *weakest* class corresponding to figure F and write $weak(\mathsf{Fig})$. As an example, consider the class of triangles ($\mathcal{T}$), isosceles triangles ($\mathcal{I}$), and equilateral triangles ($\mathcal{E}$), it is clear $\mathcal{E} \sqsubseteq \mathcal{I} \sqsubseteq \mathcal{T}$ as $\mathcal{E}$ contains the most information and is thus the strongest class.

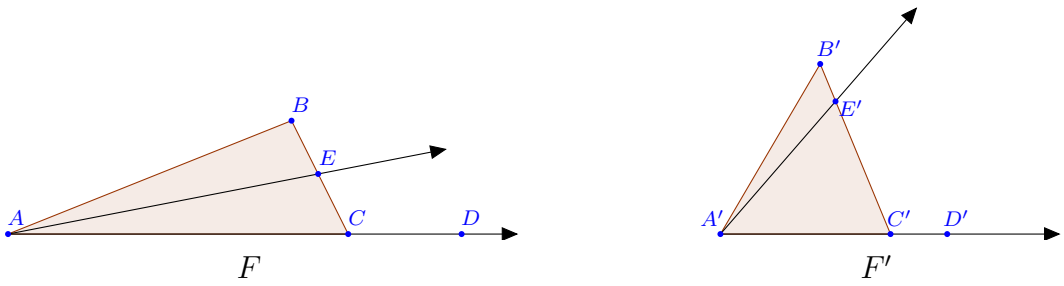


Figure 3.2: Invariant Figures Fig and Fig′ (Fig ≈ Fig′)

Given a geometric figure Fig, we will construct two sets of properties that describe Fig. The first set of properties, $\mathcal{I}(\mathsf{Fig})$, describe the invariant characteristics of Fig. That is, we note the relationships among the points, lines, and shapes that are independent of specific information about Fig; that is, angles and distances between points may differ but not the overall structure of the figures as previously described in §3.2 according to ordered geometry [27]. In Figure 3.2 two distinct figures Fig and Fig′ are invariant. We now formally define the notion of invariance between figures.

Definition 16 (Invariant Figures)**.** *Two figures G and G' are invariant if there exists a class of geometric figures* Fig *such that* $weak\,(G) = \mathsf{Fig} = weak\,(G')$*. We write* $G \approx_{\mathsf{Fig}} G'$ *to say figure G is invariant to figure G' with respect to class* Fig.

For a figure Fig, we define the theory of Fig denoted by $Th\,(\mathsf{Fig})$ to be $Th\,(\mathsf{Fig})$ where $\mathsf{Fig} = strong\,(\mathsf{Fig})$. We may also extend to the theory of a class Fig being defined as all figures in Fig implying all properties of Fig.

As an example, let figure Fig be a right triangle $\triangle ABC$ with $m\angle BAC = 90^o$ ($m\angle BAC$ refers to the measure of $\angle BAC$). The theory of Fig, denoted by $Th\,(\mathsf{Fig})$ is given by

$$Th\,(\mathsf{Fig}) = \{\triangle ABC, m\angle BAC = 90^o\} = \{\mathsf{Triangle}(A, B, C), \mathsf{RightTriangle}(B, A, C)\}.$$

Let $\mathcal{T}$ be the class of triangles and $\mathcal{T}_r$ be the class of right triangles, then we note for the right triangle F above, it is true that $F \in \mathcal{T}$ and $F \in \mathcal{T}_r$ with $\mathcal{T}_r \sqsubseteq \mathcal{T}$.

In general, $Th\,(\mathsf{Fig})$ denotes the minimal set of properties of the figure Fig.

Geometric Axioms. For Euclidean geometry, we assume modified versions of Euclid's original axioms [51]; these axioms are universally quantified. These axioms are stated below:

1. Segment Addition: If B is between A and C, then $AB + BC = AC$.
2. Angle Addition: If point D lies on the interior of $\angle ABC$, then $m\angle ABD + m\angle DBC = m\angle ABC$.
3. Angle Addition for Straight Angles: If $\angle ABC$ is a straight angle and D is any point not on $\overleftrightarrow{AC}$, then $m\angle ABD + m\angle DBC + m\angle ABC = 180^o$.
4. Algebraic Properties of equality including addition, subtraction, multiplication, and division.
5. Equality (=). congruence ($\cong$), and similarity ($\sim$) are equivalence relations.

The set of axioms describing algebraic properties of equality including addition, subtraction, multiplication, and division and those describing the fact that equality (=). congruence ($\cong$), and similarity ($\sim$) are equivalence relations are called the *algebraic axioms* and are denoted by Axm_a. In addition, a few existentially quantified axioms are assumed:

1. A line contains at least two points.
2. Through any two points, there exists exactly one line.
3. If two parallel lines are cut by a transversal, then corresponding angles are congruent.
4. SSS, SAS, and ASA congruency of triangles.
5. Corresponding Parts of Congruent Triangles are Congruent (CPCTC).
6. AA Similarity of Triangles.

Each of the axioms above requires an encoding into a logical form. For the Segment Addition Axiom to be applied, we require two distinct pieces of information: (1) three points are collinear, (2) which of the three points lies between the other two points. Consider a segment $\overline{\chi_1\chi_2}$ with point χ_3 between χ_1 and χ_2. Then

$$\mathsf{Collinear}\,(\chi_1, \chi_2, \chi_3) \wedge \mathsf{Between}\,(\chi_3, \overline{\chi_1\chi_2}) \Rightarrow \chi_1\chi_3 + \chi_3\chi_2 = \chi_1\chi_2.$$

With CPCTC, we require two congruent triangles and the labeling of the respective vertices of the congruent triangles to be consistent:

$$(\Delta ABC) \wedge (\Delta DEF) \wedge (\Delta ABC \cong \Delta DEF) \Rightarrow (\overline{AB} \cong \overline{DE}) \wedge (\angle ABC \cong \angle DEF) \wedge$$
$$(\overline{CA} \cong \overline{FD}) \wedge (\angle BCA \cong \angle EFD) \wedge$$
$$(\overline{BC} \cong \overline{EF}) \wedge (\angle CAB \cong \angle FDE)$$

Definitions of Geometric Terms. We presume standard definitions of common geometric terms; e.g.:

- *Collinear* refers to a set of points lying on one line.
- *Midpoint* of a segment refers to the point that divides a given segment into two congruent segments.

These definitions have ramifications because they imply more properties regarding a figure. For example, if M is the midpoint of $\overline{XY}$, then the definition states $XM = MY$. However, the definitions are implicit in the theory of a figure F as well as the theory of given information. For a figure Fig, we call this information the *theory of assumptions*, $Th\left(I^{\mathsf{Fig}}\right)$.

3.3.3 Synthesis Hypergraphs and Problems

The formal framework we use to represent a geometric figure together with the assumptions is a hypergraph. Proof problems will be synthesized by exploring this hypergraph. Given only a figure, we may construct a corresponding hypergraph based solely on the implicit properties, axioms, and student knowledge base. The student knowledge base comprises the lemmas and theorems that the student possesses in their knowledge base.

Definition 17 (Basic Geometry Synthesis Hypergraph)**.** *Given a figure* Fig*, the* basic geometry synthesis hypergraph *corresponding to* Fig *is* $H_b^{\mathsf{Fig}}(P, E)$ *where* P *is the set of nodes and* E *is the set of hyperedges. We define the set of nodes in the hypergraph* $P = Th(\mathsf{Fig}) \cup Th(\mathsf{Axm}_x) \cup Th(K)$ *where* $\mathsf{Fig} = strong(\mathsf{Fig})$*,* K *is the student knowledge base,*

and Axm_x *is the set of Euclid's axioms. The hyperedges* E *of the hypergraph are defined as a set of functions mapping a set of nodes to a single node:* $E \subseteq \bigcup_{i=1}^{|P|} P^i \rightarrow P$ *where* $\langle p_1, \ldots, p_\ell \rangle \rightarrow p \in E$ *if* $Th\left(\mathsf{Fig}\right) \cup Th\left(\mathsf{Axm}_x\right) \models p_1 \wedge \ldots \wedge p_\ell \Rightarrow p$ *holds true.*

Each node in the basic hypergraph is *typed* so it belongs to one discrete class in the set of types $\tau = \{algebraic, geometric\}$. We make these distinctions among nodes so that later we may formally define a problem with respect to a basic geometry hypergraph. We now define how the type of each node in a basic geometry hypergraph is acquired.

Definition 18 (Algebraic and Geometric Nodes)**.** *Let* n *be a node in a basic geometry synthesis hypergraph* H*. If* n *is a propositional formula associated with some* $a \in \mathsf{Axm}_a$*, we say* n *is a* purely algebraic node*. We define leaves* $\left(H^T\right)$ *to be the set of all nodes in* H^T *without parents. If for all* $\ell \in leaves\left(H^T\right)$ *such that there exists a path from* n *to* ℓ *in* H^T*,* ℓ *is a purely algebraic node, we say* n *is an* algebraic node*. We note that purely algebraic nodes are considered algebraic nodes. We similarly define the terms* purely geometric nodes *and* geometric nodes *for Euclid's axioms,* A_x*.*

We can extend our notion of the basic geometry hypergraph H_b^{Fig} for a geometric figure Fig by including the problem statement in the corresponding hypergraph. This is accomplished by incorporating the assumptions, I^{Fig}, and the goal, g.

Definition 19 (Standard Geometry Synthesis Hypergraph)**.** *Given a figure* Fig *and corresponding basic geometry synthesis hypergraph,* $H_b^{\mathsf{Fig}}\left(P, E\right)$*, the* standard geometry synthesis hypergraph *corresponding to* Fig *with assumptions* I^{Fig} *and goal* g*, is given by* $H_s^{\mathsf{Fig}}\left(H_b^F, P_g, E_g, g\right)$*. We define the additional set of typed nodes in the hypergraph* $P_g = Th\left(I^{\mathsf{Fig}}\right) \cup \{g\}$*. The corresponding additional hyperedges,* E_g *are a result of the theories derived from all typed nodes given by* $P \cup P_g$ *where* P *are the typed nodes defined in* H_b^{Fig}*.*

It is clear that for a figure Fig, H_b^{Fig} is a sub-hypergraph [13] of H_s^{Fig}.

If we do not distinguish between a basic hypergraph or standard hypergraph we will refer to a *problem hypergraph* (or simply hypergraph when context is clear), $H\left(P, E\right)$ where

P is the set of typed nodes and E is the set of hyperedges. We note that this definition is analogous to the logical geometry hypergraph of Definition 14.

Geometry Problems. Now that we have defined a hypergraph in the deductive space of Euclidean geometry, we can define a geometry problem with respect to problem hypergraphs. A traditional high school geometry problem in simplest form is a natural language statement, but more common is the combination of a description composed of mathematical relationships and natural language which describe a figure. In a problem hypergraph, we informally define a problem as a set of typed nodes that describe the assumptions of the problem and a corresponding typed goal node that follows from the assumptions. The corresponding path from the typed assumption nodes to the typed goal node is a solution to the problem (i.e., a proof of the goal).

Definition 20 (Basic and Standard Problems)**.** *Given a basic hypergraph H_b^{Fig} corresponding to a figure* Fig*, a* basic geometry problem *P is a statement of the form $p_1 \wedge \ldots \wedge p_k \vdash p$ for some $k > 0$ where for all i, p_i is the propositional formula corresponding to typed node n_i of H_b^{Fig}, p is the propositional formula corresponding to typed node n of H_b^{Fig}, and there exists a path $\mathcal{P}$ from $\langle n_1, \ldots, n_k \rangle$ to n. The path $\mathcal{P}$ is a* solution *to geometry problem P. For a node g, we say that $\mathscr{P}_g$ defines the collection of all paths in hypergraph H_b^{Fig} with goal node g; a* valid student solution *is any path in $\mathscr{P}_g$. A* standard geometry problem *is defined similarly for a standard hypergraph H_s^{Fig}.*

We will use the general term *problem* in situations where the context is clear. For a goal g and a set of source nodes S in a standard hypergraph $H_s^{\mathsf{Fig}}\left(H_b^{\mathsf{Fig}}, P_g, E_g, g\right)$ corresponding to figure Fig with assumptions A, we say that S is *strict* with respect to g if $S \vdash g$ is a problem and no $U \vdash g$ is a problem for $U \subset S$ as stated in Definition 12.

As mentioned in §3.1, not all problems are interesting. Interesting problems for a figure and a set of assumptions are those that require at least one or more of the assumptions, the assumptions are minimal with respect to the goal, and the goal cannot be derived from a set of algebraic expressions through purely algebraic manipulation.

Analogous Problems. We use the term *analogous* to define a problem as an independent, 'interesting' problem that mimics the difficulty and length of a given problem. For a problem P in a hypergraph H, the problem hypergraph $\tilde{P}$ is the sub-hypergraph of H induced by P. We begin with a strict view of problem analogy that views problem hypergraphs as graphs.

Definition 21 (Coarse Problem Homomorphism)**.** *Let $H\,(V, E)$ and $H'\,(V', E')$ be problem hypergraphs. Then $\phi : H \to H'$ is a* coarse problem homomorphism *if $v_i \in V$ for $1 \leq i \leq k$, for all $\langle v_1, \ldots, v_k\rangle = \vec{v} \in P(V)$ such that $(\vec{v} \to v) \in E$ for $v \in V$,*

- *v and $\phi(v)$ are typed nodes in which type $(v) =$ type $(\phi(v)) \in \tau$,*
- *$\vec{v}$ and $\phi(\vec{v})$ are sets of typed nodes in which $|\vec{v}|_t = |\phi(\vec{v})|_t$ for each type $t \in \tau$, and*
- *there exists an edge $\phi(\vec{v}) \to \phi(\{v\}) \in E'$.*

In Definition 21 we define analogous problems by requiring (1) corresponding node types be equivalent for each problem, (2) for each corresponding edge the number of source nodes of each type are equivalent and the the target node of the edge is of the same type, and (3) each edge has a corresponding edge in both problems. We may now define a coarse problem isomorphism based on the structural requirements of the coarse problem homomorphism.

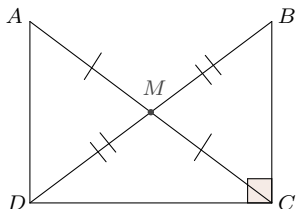
Definition 22 (Coarse Problem Isomorphism)**.** *ϕ is a* coarse problem isomorphism *if (i) ϕ is a bijection, (ii) ϕ is a coarse problem homomorphism, and (iii) ϕ^{-1} is a coarse problem homomorphism. If ϕ is a coarse problem isomorphism between H and H', we may write $H \cong_c H'$.*

Definition 23 (Coarsely Analogous Problem)**.** *Two problems P_1 and P_2 are* coarsely analogous *if there exists a coarse problem isomorphism between $\tilde{P}_1$ and $\tilde{P}_2$.*

In Figure 3.3, the two problems proving that (F) ΔBMC is isosceles and (G) ΔDMA is isosceles are coarsely analogous. However, coarse analogy can be too strong a concept

In the figure at right, assume

1. M is the midpoint of $\overline{AC}$,
2. M is the midpoint of $\overline{BD}$, and
3. $m\angle BCD = 90^o$.[75]



With the given set of assumptions using the associated figure, we may prove the following set of facts.

(A) $\Delta BMC \cong \Delta DMA$,
(B) $m\angle ADC = 90^o$,
(C) $\Delta ADC \cong \Delta BCD$,
(D) $2BM = AC$,
(E) ΔDMC is isosceles,
(F) ΔBMC is isosceles,
(G) ΔDMA is isosceles,
(H) $\overline{BC} \parallel \overline{AD}$. ($\overline{BC}$ is parallel to $\overline{AD}$.)

Figure 3.3: Provable Facts From A Geometric Problem Statement

to formally capture the notion of "analogy". For example, in Figure 3.3 a student who has been able to prove statements (F) and (G) should also be able to prove statement (E) since all three statements require one to prove that a particular triangle is isosceles although the task of proving (E) is not coarsely analogous to that of proving (F) nor (G).

Formally capturing a weaker notion of analogy motivates the following definition.

Definition 24 (Goal Analogous Problems)**.** *Let P_1 and P_2 be two problems with goals g_1 and g_2, respectively. We say problems P_1 and P_2 are* goal analogous problems *if type $(g_1) =$ type (g_2) and strong $(g_1) =$ strong (g_2). This is clearly an equivalence relation and we refer to the induced equivalence classes as a* goal analogous partition*.*

3.4 Algorithm for Problem Generation

Our algorithm for problem generation has three steps. The first step creates a hypergraph according to Definition 14 that represents all possible proofs for all possible problems over a given pair of a user-provided figure and axioms. The second step systematically enumerates all minimal sets of assumptions (Algorithm 3.1). The third step enumerates, for each minimal set of assumptions I, all possible goal sets G such that (I, G) is a strictly

Using the provided figure and the fact that M is the midpoint of $\overline{AB}$, prove that $2AM = AB$ and $2MB = AB$.

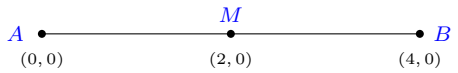
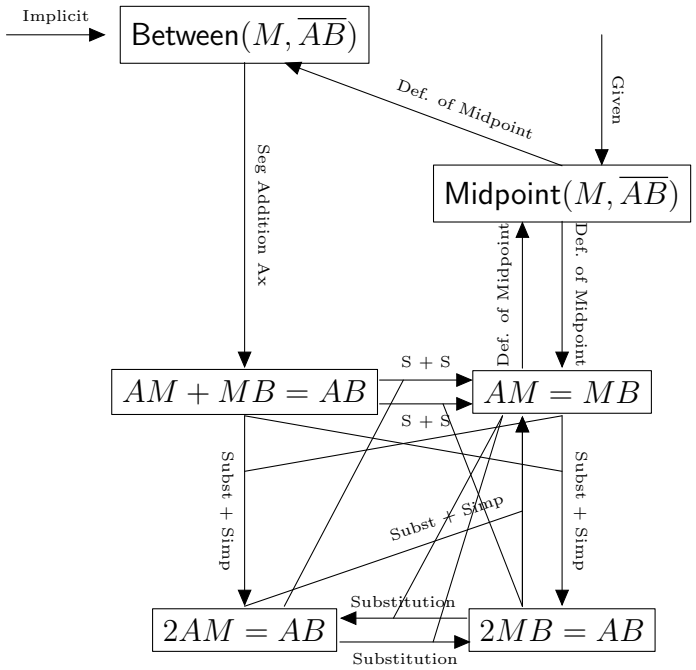


Figure 3.4: Statement of the Midpoint Theorem



("S + S" refers to algebraic substitution followed by algebraic simplification.)

Figure 3.5: Logical Geometry Hypergraph for the Midpoint Theorem of Figure 3.4

interesting problem.

In the following exposition, we focus on clarity rather than performance. The enumeration of problems is exponential in the worst case; we show in §3.6 that nevertheless, the enumeration can be performed successfully in practice.

3.4.1 Step 1: Hypergraph Construction

We compute a logical geometry hypergraph as defined in Definition 14. The input to the algorithm is a geometry figure Fig drawn to scale and a set of axioms represented as Horn clauses. The algorithm internally computes the sets $\mathcal{I}(\mathsf{Fig})$ and $\mathcal{E}(\mathsf{Fig})$ and then constructs the logical geometry hypergraph for $(\mathsf{Fig}, \mathsf{Axm})$. The hypergraph is used to

Algorithm 3.1 Algorithm *AllMinimalSets*

Input: Figure Fig, axioms Axm
Output: Set of all minimal sets of $\mathcal{E}(\mathsf{Fig})$
AllSets $\leftarrow \{\emptyset\}$
Old $\leftarrow \emptyset$
while *AllSets* $\neq$ *Old* **do**
    Old $\leftarrow$ *AllSets*
    for all $I \in$ *AllSets* **do**
        for all $f \in \mathcal{E}(\mathsf{Fig})$ s.t. *Derive*$(I) \neq$ *Derive*$(I \cup \{f\})$ **do**
            AllSets $\leftarrow$ *AllSets* $\cup \{I \cup \{f\}\}$
        end for
    end for
end while return *AllSets*

compute *Derive*(T) queries for sets $T \subseteq \mathcal{E}(\mathsf{Fig})$ in the subsequent steps of the algorithm.

As an example, we consider the Midpoint Theorem, often the first proof in a geometry course, as stated in Figure 3.4. We note that the statement of the Midpoint Theorem has $I = \{\mathsf{Midpoint}(M, \overline{AB})\}$ and $G = \{2AM = AB, 2MB = AB\}$ with $|G| = 2$. We also note that the figure associated with the problem in Figure 3.4 provides a set of sample coordinates demonstrating the embedding of the figure in the Euclidean plane thus facilitating computation of the geometric facts of $\mathcal{I}(\mathsf{Fig})$ and $\mathcal{E}(\mathsf{Fig})$.

We then construct the logical geometry hypergraph corresponding to the problem in Figure 3.4 in Figure 3.5. In this construction of the hypergraph for Figure 3.4, the geometry facts describing each node are self-explanatory except for $\mathsf{Between}(M, \overline{AB})$. The Between predicate construct (1) implies collinearity of the three points M, A, and B and (2) M falls between the endpoints of the segment A and B. In other words, for $M \neq A$ and $M \neq B$, $\mathsf{Between}(M, \overline{AB}) \iff AM + MB = AB$.

3.4.2 Step 2: Minimal Assumption Generation

A set $T \subseteq \mathcal{E}(\mathsf{Fig})$ is *minimal* if either $T = \emptyset$ or for each $t \in T$, we have that $T \setminus \{t\}$ is minimal and *Derive*$(T) \neq$ *Derive*$(T \setminus \{t\})$. Minimality is a necessary condition for an interesting problem.

Algorithm 3.2 Algorithm *GenProblem*

```
1: Input: Figure Fig, axioms Axm, minimal set I ⊆ E(Fig)
2: Output: Strictly interesting problem (I, G)
3: G = ∅
4: while ∃f ∈ I s.t. G ⊆ Derive(I \ {f}) do
5:     f = choose({f ∈ I | G ⊆ Derive(I \ {f})})
6:     T = Derive(I) \ Derive(I \ {f})
7:     g = choose(T \ I)
8:     G = G ∪ {g}
9: end while
10: return (I, G)
```

In the second step, the problem synthesis algorithm systematically enumerates all minimal sets of assumptions; Algorithm 3.1 is a simple fixed-point procedure to compute the set of all minimal sets.

Theorem 3.4.1 (Completeness of *AllMinimalSets*)**.** *AllMinimalSets*(Fig, Axm) *returns the set of all minimal sets for a pair* (Fig, Axm)*, .*

Proof. As a base case, consider a singleton fact $p \in \mathcal{E}(\mathsf{Fig})$ that defines a minimal set $\{p\}$. On Line 3, $\mathit{AllSets} = \{\emptyset\}$. Therefore, the first time through the generative loop from Line 8 through Line 10, $I = \emptyset$. Hence, for $p \in \mathcal{E}(\mathsf{Fig})$, since $\{p\}$ is a minimal set, $\emptyset \cup \{p\} = \{p\}$ is added to *AllSets* on Line 9 and is thus generated by *AllMinimalSets*.

Suppose $M = \{p_1, \ldots, p_k\}$ is a minimal set containing $k > 1$ facts generated by *AllMinimalSets*. Also suppose for some $q \in \mathcal{E}(\mathsf{Fig})$, $M \cup \{q\}$ is a minimal set. As M is a minimal set, $M \in \mathit{AllSets}$. Hence, at some point during execution, $I = M$ (Line 7). Since $q \in \mathcal{E}(\mathsf{Fig})$ and $M \cup \{q\}$ is a minimal set containing $k+1$ facts satisfying the condition on Line 8, it follows $M \cup \{q\} \in \mathit{AllSets}$ (Line 9). □

3.4.3 Step 3: Strictly Interesting Problem Synthesis

The final step enumerates, for each minimal set of assumptions I, all possible goal sets G such that (I, G) is a strictly interesting problem.

We present the third step as the non-deterministic procedure in Algorithm 3.2. It takes as input a figure Fig and axioms Axm, as well as a minimal set I of explicit predicates. It computes a strictly interesting problem by "growing" a set G of goals and returns (I, G) as the generated problem. Initially, the set G is empty (Line 3). While the current set of goals is not strong enough to ensure the problem is interesting (Line 4), the algorithm generates a new goal. To generate a new goal, the algorithm finds (non-deterministically, Line 5) an assumption f that is not used to prove the current set of goals and finds (non-deterministically, Line 6) a goal that is derivable using I but not without this assumption. Notice that since I is minimal, the set T on Line 6 is non-empty. However, to ensure the condition $I \cap G = \emptyset$, we choose g from the set $T \setminus I$ on Line 7, which may be empty.

By construction, Algorithm 3.2 ensures that returned problems are strictly interesting. For the returned pair (I, G), since the *while* loop exits, we know that every $f \in I$ is necessary to prove some goal in G; hence (I, G) is interesting. Further, the problem is strictly interesting since the algorithm returns a minimal set of goals G.

The non-deterministic choices of the algorithm are denoted by the *choose* operator, which selects an element of a set (if non-empty), and fails otherwise. By iterating over possible non-deterministic choice, the algorithm can generate every possible strictly interesting problem with assumption I.

Finally, in order to *generate a complete problem*, we can check that the input I to procedure *GenProblem* can derive all explicit facts, i.e., $\mathit{Derive}(I) = \mathcal{E}(\mathsf{Fig})$.

Theorem 3.4.2 (Soundness of *GenProblem*)**.** *If* $\mathit{GenProblem}(\mathsf{Fig}, \mathsf{Axm}, I)$ *returns* (I, G) *for a minimal set* I*, then* (I, G) *is a strictly interesting problem over* $(\mathsf{Fig}, \mathsf{Axm})$*.*

Proof. Suppose *GenProblem* returns (I, G) where $G = \{g_1\}$. In this base case it is clear that executing *GenProblem* on Line 3 that $\emptyset \subset G$. $(I, \emptyset)$ is clearly not an interesting problem and the loop (Line 4 to Line 9) will be executed. With a choice of g_1 (Line 7) *GenProblem* will generate $(I, \{g_1\})$. It follows that $(I, \{g_1\})$ is strictly interesting since the only strict subset of $\{G_1\}$ does not result in an interesting problem.

Now suppose *GenProblem* returns (I, G) where $G = \{g_1, \ldots, g_k\}$ for $k > 1$. Take some $g \in G$. During execution, the loop condition (Line 4) in *GenProblem* would be satisfied for $G \setminus \{g\}$; thus $(I, G \setminus \{g\})$ would not be returned as a strictly interesting problem. That is, there exists $f \in I$ such that $G \setminus \{g\} \subseteq \mathit{Derive}(I \setminus \{f\})$. Specifically, for all subsets $G' = G \setminus \{g\}$ where g is an arbitrary element of G, *GenProblem* would continue to loop since (I, G') is not an interesting problem. The final loop execution would non-deterministically choose g (Line 7) to construct the original set $G = G' \cup \{g\} = \{g_1, \ldots, g_k\}$ for $k > 1$. It follows (I, G) is a strictly interesting problem. □

Theorem 3.4.3 (Completeness of *GenProblem*)**.** *If* (I, G) *is a strictly interesting problem for* $(\mathsf{Fig}, \mathsf{Axm})$*, there is a run of* $\mathit{GenProblem}(\mathsf{Fig}, \mathsf{Axm}, I)$ *that returns* (I, G)*.*

Proof. Suppose (I, G) is a strictly interesting problem for $(\mathsf{Fig}, \mathsf{Axm})$ where $G = \{g_1, \ldots, g_k\}$ for $k \geq 1$. We will construct a set of G' from $\emptyset$ until $G' = G$ at the end of the run.

In the first execution of the loop, we have for all $f \in I$, $\emptyset = G' \subseteq \mathit{Derive}(I \setminus \{f\})$. We choose some f and subsequently some $g_c \in \mathit{Derive}(I) \setminus \mathit{Derive}(I \setminus \{f\}) \setminus I$ (Line 7). We note that many facts may exist in $\mathit{Derive}(I) \setminus \mathit{Derive}(I \setminus \{f\}) \setminus I$; however, we non-determindistically choose a desired goal in the G: $g_c \in G$. Put $G' = \{g_c\}$. Since (I, G) is strictly interesting, (I, G') is not interesting and looping continues. If G is a singleton set, looping would cease and *GenProblem* would successfully generate (I, G).

Suppose $G' \subset G$ where $G' = \{g_1, \ldots, g_{k-1}\}$ for $k > 1$ is constructed while executing *GenProblem*. Since (I, G') is not interesting, looping continues and we non-deterministically choose f for which $g_k \in \mathit{Derive}(I) \setminus \mathit{Derive}(I \setminus \{f\}) \setminus I$. Hence, on Line 8, $G = G' \cup \{g_k\}$. We have successfully constructed (I, G) as a strictly interesting problem; looping will cease and the desired problem will be returned. □

Figure 3.1 shows some problems that were automatically generated by our algorithm. Figure 3.6 is the minimal solution to the stated problem as derived by *GeoTutor*.

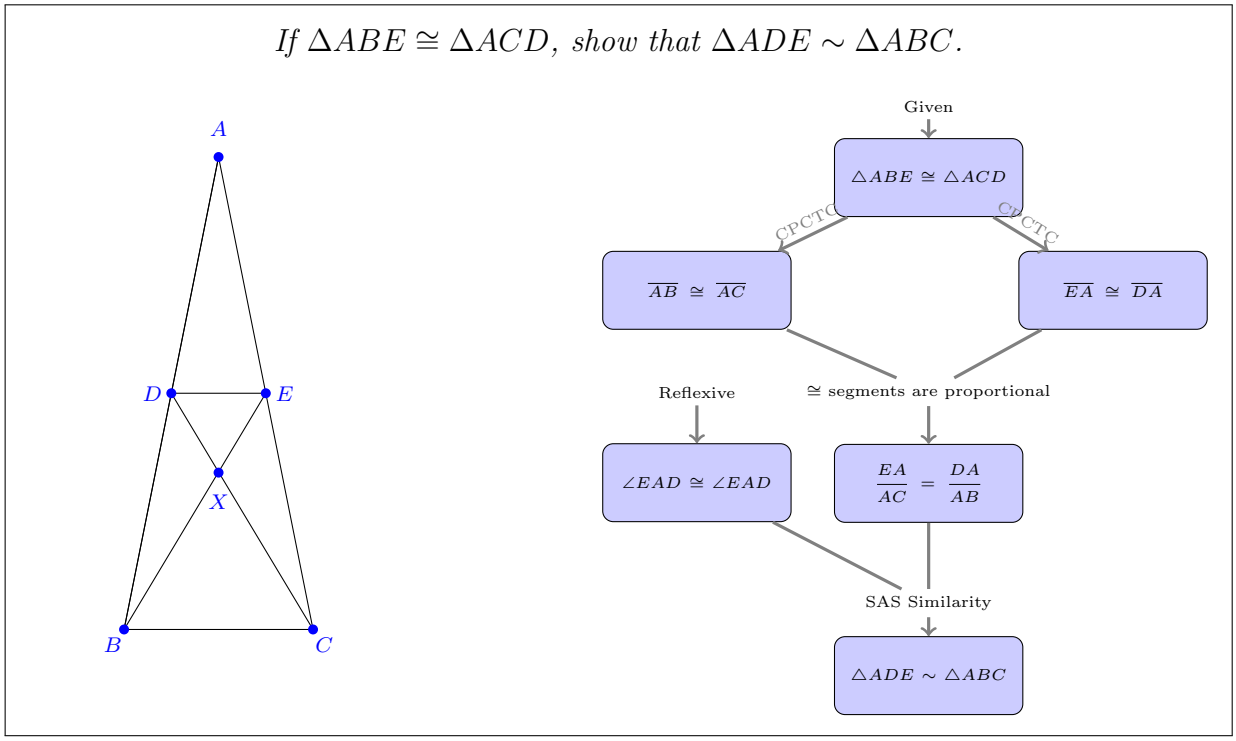


Figure 3.6: Example Problem and Minimal Solution Generated by *GeoTutor*

3.5 Problem Generation Interface

Before we present our problem synthesis algorithm, we provide a user's view to interacting with our system: how the user may interact with our system to obtain a set of desired problems. The user provides a geometry figure drawn to scale and a set of axioms as inputs to the system, and can specify parameters to generate a desired set of problems with specific features.

3.5.1 Features of a Geometry Problem

A geometry problem $P = (I, G)$ over a pair $(\mathsf{Fig}, \mathsf{Axm})$ has several features such as:

- The objects of the figure Fig and their properties $\mathcal{I}(\mathsf{Fig})$, $\mathcal{E}(\mathsf{Fig})$, e.g., the number of points, triangles, etc.
- The size of the goal set $|G|$.
- The type of the goal, e.g., congruent triangles, equal segments, etc.
- Quantitative features of a proof such as depth of a proof (i.e., the longest path from

the assumptions to the goal in the proof), the width of a proof (maximal number of nodes in a level in the proof), the number of deduction steps (i.e., the number of hyperedges in the proof), and the number of axioms used. These features can be computed from the representation of proofs in the hypergraph.

- A subset of Axm that occurs in every proof of the problem.
- Whether the problem is complete or not.

Our system allows defining arbitrary features as long as they are efficiently computable from the syntactic description of the problem or from the hypergraph representation.

3.5.2 Query Interface to Problem Generation

We propose an interface where the teacher can specify a relational query over the set of problem features and obtain a corresponding set of problems. We describe a semi-automated methodology to support this interface. Our methodology requires manual input of $(\mathsf{Fig}, \mathsf{Axm})$ pairs. For each such pair, we generate the set of all interesting problems using the problem generation technique described in §3.4. This set of problems, along with their features, populate a relational database. We may then query the database using a standard relational query (§3.6 gives examples of such queries with results in Table 3.2).

A student or teacher may define their own pair $(\mathsf{Fig}, \mathsf{Axm})$ using their own creativity or directly from textbooks to generate fresh problems corresponding to that pair. In that respect, our methodology has a multiplicative effect: starting from the figure of a problem, our algorithms generate many more problems over the same figure.

3.6 Experimental Results

We first describe our benchmark set of problems and characteristics of the corresponding figures. Second, we evaluate our solution technique with respect to time required to construct the corresponding hypergraph and identify the solution path. Last, we correlate structural characteristics of a solution with respect to the time taken to generate that solution.

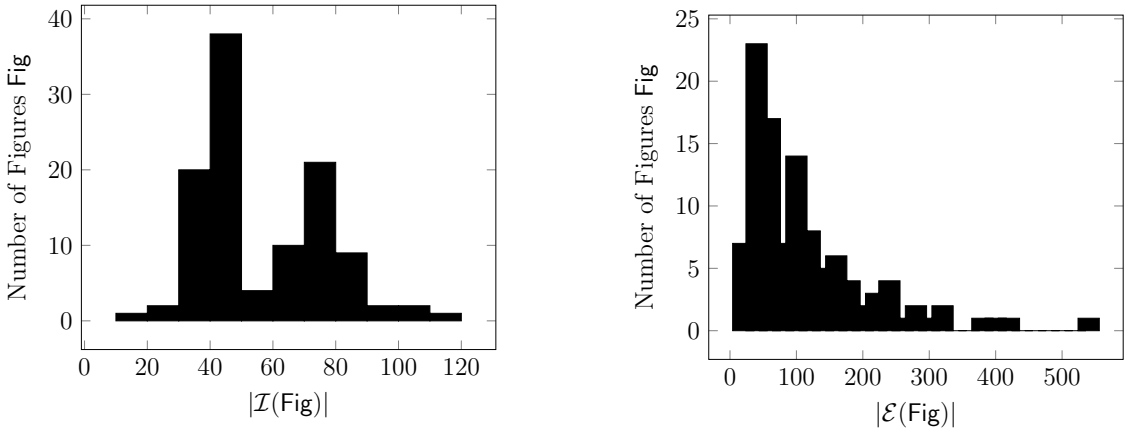


Figure 3.7: Histogram of $|\mathcal{I}(\mathsf{Fig})|$ and $|\mathcal{E}(\mathsf{Fig})|$ Per Figure F

3.6.1 Benchmark

We ran our problem generation algorithm on a set of 110 figures taken from standard mathematics textbooks in India [76, 75] as well as textbooks and workbooks popular in the United States [16, 56, 64, 51]. We used a uniform set of axioms for all of our experiments; this set included axioms related to parallel lines, congruent triangles, similar triangles, etc.

The distribution of these 110 figures described by the number of the implicit facts per figure, $|\mathcal{I}(\mathsf{Fig})|$, is a bimodal distribution with modes around 40 and 75 and mean 46.5 as shown in Figure 3.7. The bimodal distribution indicates our attempt to balance our experiments with simple as well as more complex figures.

The distribution of these 110 figures described by the number of deduced facts per figure, $|\mathcal{E}(\mathsf{Fig})|$, is a positively-skewed distribution as shown in Figure 3.7 with mean 108, median 82, and standard deviation 96.7. The skewed distribution indicates how few figures result in a large hypergraph making our problem generation algorithm often run efficiently in practice.

3.6.2 Evaluation of Algorithm *GenProblem*

We now present evaluation of our problem generation algorithm *GenProblem* with respect to the number of problems that it generates as well as the time taken to generate

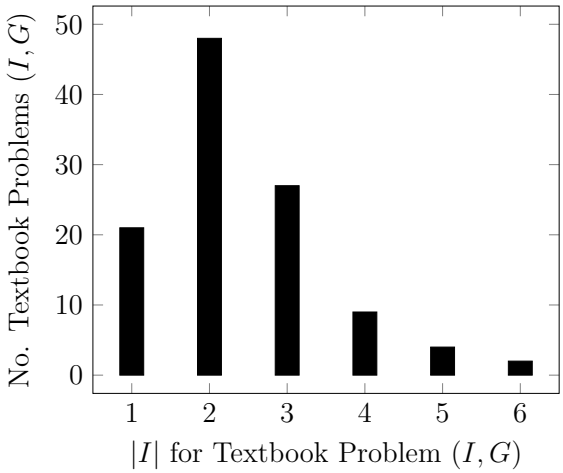
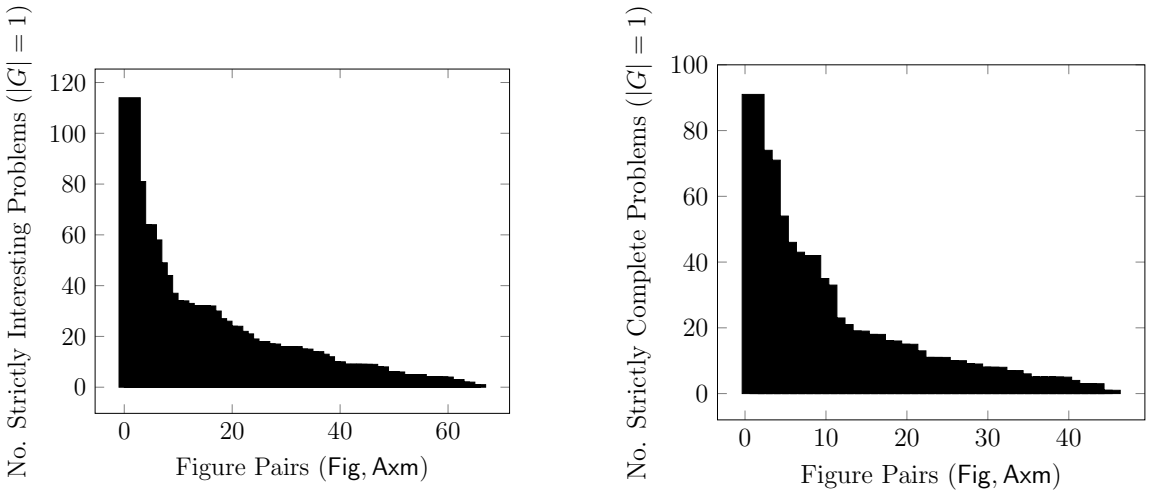


Figure 3.8: Number of Assumptions $|I|$ Per Textbook Problem (I, G)

those problems. We ran our experiments on a laptop with Intel Core i5-2520M CPU at 2.5GHz with 8 GB RAM on 64-bit Windows 7 operating system.

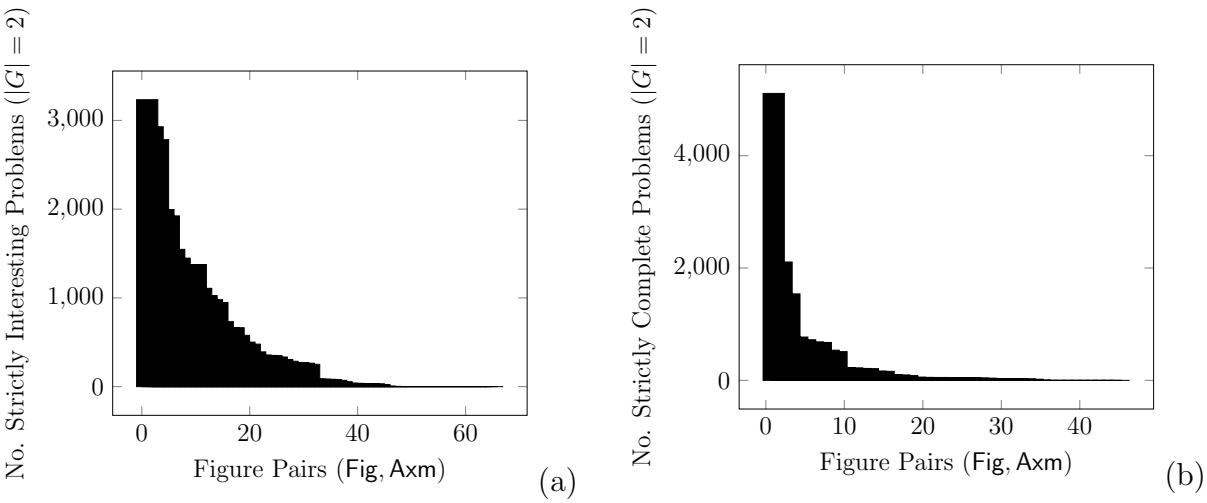
We modified *GenProblem* to only generate problems where $|G| \leq 2$. This is because our preliminary prototype encountered memory issues with $|G| > 2$ since the problem generation procedure is exponential in $|G|$. For each $(\mathsf{Fig}, \mathsf{Axm})$ pair, we fixed I to be the minimal set of assumptions that corresponded to the original textbook problem description corresponding to the figure F. For the 110 figures we observed a mean of 2.3 assumptions per figure with standard deviation 1.1; Figure 3.8 presents statistics on the size of this fixed minimal set per figure.

We found that complexity of the figure correlates with the length of time to process: more implicit facts in a figure results in more explicit facts and thus requires more time to generate problems. Given a set of assumptions I over a pair $(\mathsf{Fig}, \mathsf{Axm})$, we determine the Boolean classification whether I completely defines Fig. We may informally describe a complete problem as a problem that is not open to interpretation. That is, complete problems are ideal for formal assessments. On the other hand, interesting problems are more malleable and therefore more applicable to homework or in-class investigations. Textbook problems are generally a mix of interesting and complete problems. We found for only 45



(a) (b)

Figure 3.9: Strictly Interesting (a) and Complete (b) Problems Generated Per Pair (Fig, Axm) ($|G| = 1$)



(a) (b)

Figure 3.10: Strictly Interesting (a) and Complete (b) Problems Generated Per Pair (Fig, Axm) ($|G| = 2$)

of 110 figures, the original textbook problem associated with it was complete. We expected a larger number of complete problems, but found that when drawing figures into our front-end slate, we were more likely to construct figures with unintended facts (e.g. points were likely to be midpoints, triangles likely to be isosceles or equilateral). This psychological factor lead to a greater number of original textbook problems being classified as interesting (but not complete).

Our methodology results in a large multiplicative effect: from a single pair $(\mathsf{Fig}, \mathsf{Axm})$ we are able to generate many problems. For the 65 of 110 original textbook figures that were classified as corresponding to interesting (but not complete) problems, we generated a total of 1309 and an average of 20.1 strictly interesting problems (I, G) where $|G| = 1$; the associated distribution is shown in Figure 3.9(a). For the remaining 45 of 110 original textbook figures, which were classified as corresponding to complete problems, we generated a total of 877 and an average of 19.5 strictly complete problems (I, G) where $|G| = 1$ with distribution in Figure 3.9(b). For $|G| = 2$, we generated 14760 strictly complete problems and 31801 strictly interesting (but not complete) problems. When $|G| = 2$ we have an empirical validation of the exponential growth in the number of generated problems. For a fixed set of assumptions I, the definition of a strict problem dictates $|I| \geq |G|$ for any G. Since many of our original textbook problems had $|I| = 1$, many figure pairs $(\mathsf{Fig}, \mathsf{Axm})$ cannot generate problems with more than a single goal. The corresponding distributions (shown in Figure 3.10) are heavily skewed with mean 489 and median 84 for strictly interesting (but not complete) problems as well as mean 328 and median 49 for strictly complete problems.

GenProblem took an average time of 4.7 seconds (with standard deviation of 10.5 seconds) per $(\mathsf{Fig}, \mathsf{Axm})$ pair to generate the above mentioned problems with $|G| \leq 2$. For a given $(\mathsf{Fig}, \mathsf{Axm})$ pair, the majority of the processing time is in construction of the saturated hypergraph. Therefore, we expect a correlation between the number of explicit facts for Fig and the amount of time to process. As the worklist construction of $H(\mathsf{Fig}, \mathsf{Axm})$ requires that we compare each newly deduced node against all existent nodes in H, we expect hypergraph construction to be quadratic in the number of nodes in H; we have a strong quadratic correlation with coefficient $r^2 = 0.7785$.

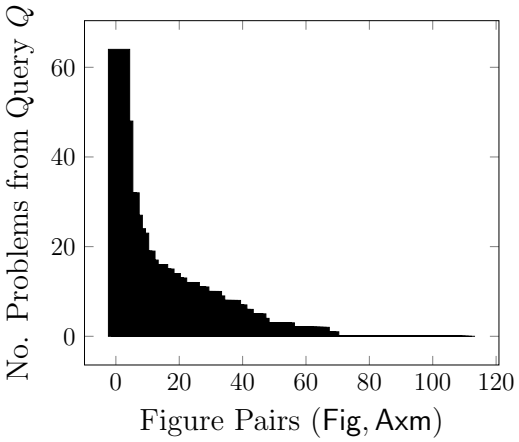
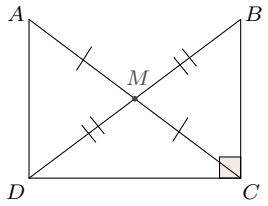


Figure 3.11: Problems Per Pair (Fig, Axm) for Query $Q = \{\text{steps} = 6 \text{ to } 10, \text{width} = 4 \text{ to } 8\}$

Let (Fig, Axm) be a pair where Fig is the figure at right and Axm is our common set of axioms.



The original problem from the textbook over (Fig, Axm) is (I, G), where

$$I = \{\mathsf{Midpoint}(M, \overline{BD}), AM = MC, \mathsf{RightAngle}(B, C, D)\}$$

and

$$G = \{\triangle BMC \cong \triangle DMA, \mathsf{RightAngle}(A, D, C), \triangle ADC \cong \triangle BCD, 2BM = AC\}.$$

The query Q generates several new problems of the form (I', g') over the pair (Fig, Axm), where $I' = I$ and g' takes on any of the following propositions.

- CD is an altitude of $\triangle ADC$
- $\mathsf{RightTriangle}(A, D, C)$
- $AD \perp CD$
- $AD \parallel BC$
- $\angle CDB$ and $\angle MAD$ are complementary

Figure 3.12: Satisfying Query Q over Fig where $Q = \{|G| = 1, s = 6 - 10, w = 4 - 8\}$

3.6.3 Effectiveness of Our Methodology

Once problems are generated from all pairs (Fig, Axm), we may obtain problems with similar features across different figures. We consider the use-cases of a teacher and student.

Table 3.2: Number of Problems (and Figures) Satisfying Queries (s: steps, d: depth, w: width, $G = \{\cong \triangle s\}$)

i	Query: Q_i	Number of Problems	Over (Fig, Axm)
1	$\{\mathrm{s} = 1-2, G\}$	23	22
2	$\{\mathrm{s} = 3-7, G\}$	73	50
3	$\{\mathrm{s} = 6, \mathrm{d} = 4, \mathrm{w} = 5, G\}$	1	1
4	$\{\mathrm{s} = 6, \mathrm{d} = 4-5, G\}$	54	28
5	$\{\mathrm{s} \geq 10, G\}$	26	14

Consider the scenario where a teacher wants to generate a set of problems for students to review before the final exam. The teacher might construct a query Q to obtain problems that are (1) medium-to-hard (6 to 10 deductive steps) with (2) average width (4 to 8), and (3) contain a single goal. Q returns a total of 706 problems from our database with an average of 6.4 problems per pair (Fig, Axm); the graph in Figure 3.11 details the number of problems per pair that satisfy Q. Figure 3.12 shows a sample of those 706 problems.

Now let's consider a common scenario for a student preparing for an exam that will test on proving triangles congruent using any technique. In this case, the student may specify a series of queries Q_i capturing problems of increasing difficulty as measured by the number or kind of deductive steps required. Each Q_i also specifies that the problem should have a single goal g that makes use of CongruentTriangles predicate. These queries Q_i are discussed below with the query results enumerated in Table 3.2.

The student begins by specifying the query $Q_0 = \{$ steps $= 1$ to $2,\ g\}$ and is provided one of the 23 problems. Assuming success with a few practice problems, the student seeks a series of more difficult problems and defines $Q_1 = \{$steps $= 3$ to $7,\ g\}$. After completing some of the 73 possible interesting problems that match Q_1, the student encounters a problem that is intriguing in its structure. As a point of interest and practice, the student defines a query based on the parameters of the problem just completed: $Q_2 = \{$steps $= 6$, depth $= 4$, width $= 5$, $g\}$. The result of the query is that there is no other problem with the defined characteristics. Instead, the student relaxes the restrictions

resulting in $Q_3 = \{\text{steps} = 6, \text{depth} = 4 \text{ to } 5, g\}$ and acquires 26 problems. Finally, the student may provide a query that requires the proof problem to have more than 10 deductions steps: $Q_4 = \{\text{steps} \geq 10,\ g\}$. After successfully completing one or more of these 26 problems, the student can be confident in their preparation for the exam.

Chapter 4
Synthesis of Problems and Solutions for Shaded Area Geometry Reasoning

We motivate and address the task of automatically solving and computing characteristics of shaded area geometry problems and formalize the notion of a shaded area geometry problem and its solution. Our approach consists of identifying atomic regions in a pixel-based geometry image, building an *analysis hypergraph* that represents all facts that can be derived of the figure (using saturation based reasoning) and then finding a path in the hypergraph from the given facts to the goal. On a corpus of 102 problems taken from popular high-school geometry textbooks, our tool *GeoShader* successfully solved and characterized all problems in an average time of 13.4 seconds.

4.1 Introduction

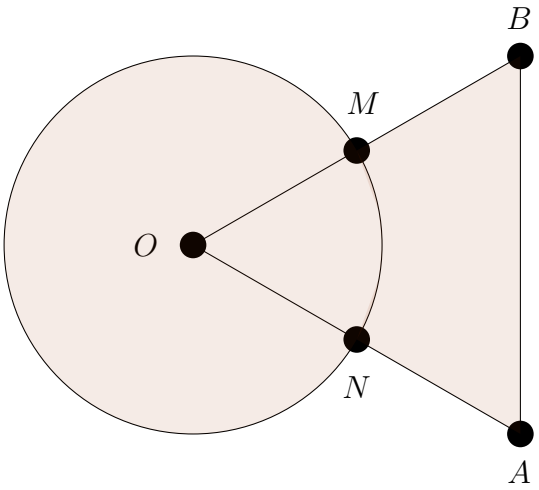
We describe *GeoShader*, a tool that can solve *shaded area* geometry problems. A shaded area problem is composed of a geometric figure, a set of given facts about that figure, and a shaded region in the figure whose area is to be found. Figure 4.1 describes a sample shaded area problem.

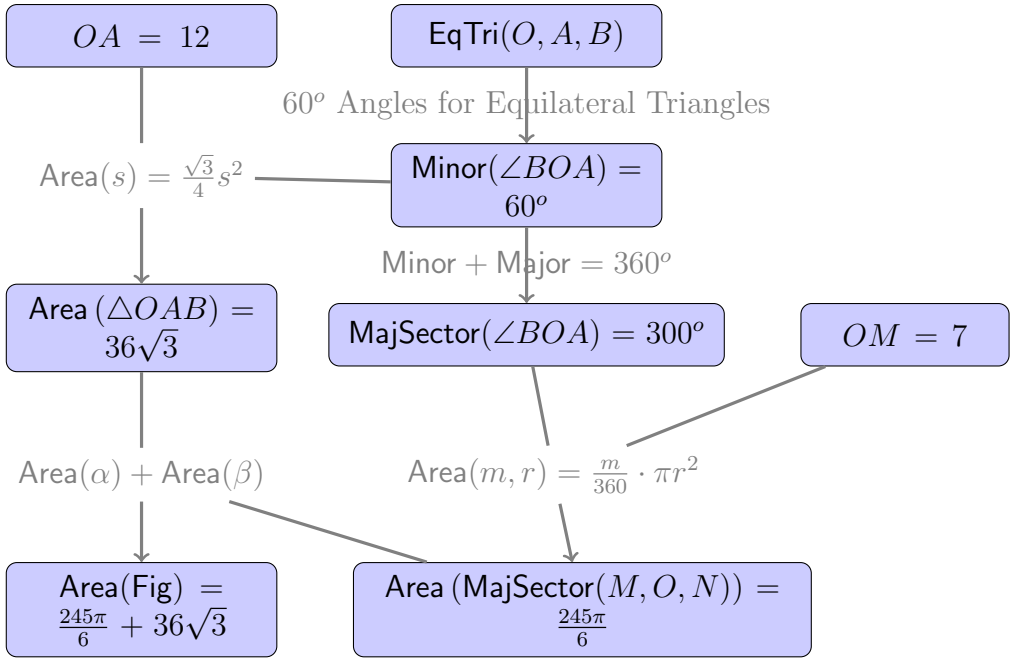
A solution to a typical high school geometry problem requires a student to use deductive logic while reasoning about the visual elements in a given figure. For this reason, geometry can be challenging for students. Shaded area problems go a step further by requiring recall of formulae for different shapes as well as exercising the associated quantitative skills necessary to compute the area of the desired region.

While a typical shaded area problem is quite demanding of a student to exercise their skills, it has a simply stated quantitative answer. This makes shaded area problems an ideal question format for multiple choice problems compared to geometry construction or proof problems that have many possible solutions and require expert knowledge to assess a solution. It becomes clear why shaded area problems are often encountered on standardized

Find the area of the shaded region where a circular arc of radius 7cm has been drawn with vertex O of an equilateral $\triangle OAB$, of side 12cm, as center. [21]

The goal region g is the entire figure with explicit facts $\{\mathsf{EqTri}(O, A, B), OM = 7, OA = 12\}$. The solution summing the areas of $\mathsf{MajSector}(M, O, N)$ and $\mathsf{EqTri}(O, A, B)$ is depicted as a hypergraph.





$$A_g = \mathsf{MajSector}(M, O, N) + \mathsf{EqTri}(O, A, B) = \tfrac{245}{6}\pi + 36\sqrt{3}$$

Figure 4.1: Example Shaded Area Problem and Solution

high school Mathematics examinations (e.g., ACT, SAT, State Comprehensive Assessment Exams [61, 69], etc.) and even on some graduate level ones (e.g., GRE quantitative).

One can represent the solution process for a shaded area problem as a directed acyclic graph (DAG), where each node represents intermediate facts that are true of the figure

(and are derivable from the predecessor nodes), and are useful for computing the goal area. A solution to a shaded area problem thus has quantifiable features corresponding to properties of its solution DAG. For example, a solution (and its corresponding problem) has depth and width. *GeoShader* computes the structural features of a solution as well as other descriptive features of a solution, including a level of difficulty corresponding to the number of deductions, geometric facts, and facts related to area. Each of these features gives a teacher the ability to effectively identify or compare problems (with associated solution) when constructing homework problem sets or exams.

GeoShader solves a shaded area problem by first dissecting the given figure into its closed, constituent areas using a planar graph-based representation. It then arranges the shapes in the figure into a hierarchy followed by a fixed-point technique to acquire the area of regions in the figure. The area of region is thus a linear combination of areas of other regions. *GeoShader* represents all possible algebraic decompositions as a hypergraph in which the solution to a problem can be obtained by traversing this hypergraph.

GeoShader can also synthesize all possible *interesting* shaded area problems from a given geometric figure. It can classify problems to be interesting (there are no irrelevant 'top'-level shapes) and complete (the areas of all regions in the figure can be computed from the assumptions).

We evaluated the effectiveness of our problem generation algorithm on 102 figures taken from popular geometry textbooks and exams. *GeoShader* solved each problem in an average of 13.4 seconds and generated an average of 257 problems.

Lastly, given a set of shapes (e.g., triangles, circles, etc.) we compose them in all possible ways (e.g., one shape inside another, one shape sharing a side with another, etc.) using a template-based approach. That is, *GeoShader* represents families of area problems as templates $\alpha_1 \pm \ldots \pm \alpha_k$, where α_i are shapes. The result from *GeoShader* is a geometric figure and the associated set of interesting shaded area problems. We evaluated our figure synthesis algorithm using a small set of polygons generating 3533 distinct figures.

This chapter makes the following contributions:

- In §4.4 we formalize the notion of a shaded area geometry problem (along with some useful features associated with it) and its solution.
- We present a technique for generating fresh figures and the associated problems (§4.5).
- We formalize the notion of an interesting and complete shaded area geometry problem (§4.7).
- We describe an algorithm to efficiently solve shaded area problems (§4.6).
- We describe algorithms to efficiently generate fresh problems from existing figures (§4.7).
- We describe experimental results illustrating the efficacy of our problem solving algorithm, problem generation algorithm, and figure synthesis techniques (§4.8).

4.2 Preprocessing: Constructing a Figure of Convex Components

Our formalization and figure analysis algorithms depend require the input shaded area figure be composed of convex elements. In this section we describe the process that all figures must go through in order to satisfy the forthcoming analyses. In the forthcoming discussion, for simplicity, we will refer to circles, sectors, and arcs as circle-based components.

4.2.1 Implicit and Computable Properties of a Figure

We begin by describing a geometric figure and then describing the constituent components of a figure. As described in Chapter 3, we consider a *geometric figure* to be a collection of immutable, geometric objects (points, lines, circles) embedded in the Euclidean plane. When context is clear, we will refer to a geometric figure as a figure.

For a figure Fig, we maintain the set of points that define the components of the figure (Fig.DefinePts) and defining characteristics for use as the basis of analyses. For circle-based components (Fig.Circles), we maintain the centers (Fig.Centers) and length each radius. For all segments, we maintain endpoints (Fig.Endpoints) and the set of explicit segments

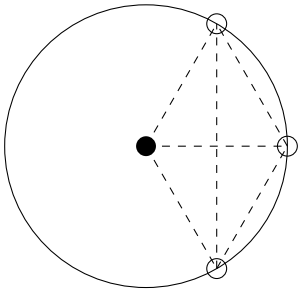


Figure 4.2: Constructing Radii and Chords

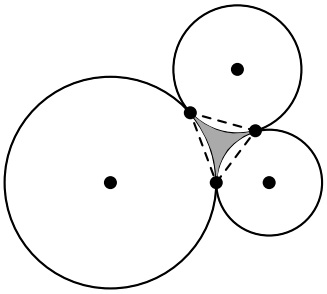


Figure 4.3: Sample Pathological Region

(Fig.eSegments); in the case of three collinear points A, M, Z where $AM + MZ = AZ$, Fig.eSegments $= \{\overline{AM}, \overline{MZ}, \overline{AZ}\}$. All other points that are labeled in Fig, but are not required in defining Fig, are maintained as Fig.Labeled. Last, for a component C of a figure Fig and a set of points P, we define $(P)\,.\mathsf{LiesOn}\,(C) \subseteq P$ such that for all $p \in (P)\,.\mathsf{LiesOn}\,(C)$, p lies on C.

In Figure 4.1, figure Fig contains Fig.Circles $= \{\mathsf{Circle}(O, OM)\}$, Fig.Centers $= \{O\}$, and Fig.eSegments $= \{\overline{AB}, \overline{OM}, \ldots\}$. We provide these defining elements of Fig as demonstrative and not exhaustive.

We use these ground facts about a figure as a basis to compute other facts, including the set of implied and extended segments.

Implied Segment Construction. A typical shaded area problem often omits implied information; for example, radii and chords in circles are often implied, but not drawn. In order to compute this set of implied segments of a figure Fig (Fig.iSegments), we define the set of intersection points as those points arising from the intersection of circle-based components and explicit segments:

$$\mathsf{Fig.inter} = \{p \mid \mathsf{Intersection}\,(c, s) \text{ for } c \in \mathsf{Fig.Circles} \text{ and } s \in \mathsf{Fig.eSegments}\}.$$

We construct the set of implied radii and chords based on the set of intersection points:

$$\mathsf{Fig.iRadii} = \bigcup_{C \in \mathsf{Fig.Circles}} \{\overline{EP} \mid \forall P \in (\mathsf{Fig.inter})\,.\mathsf{LiesOn}\,(C)\,,$$
$$E = C.\mathsf{Center}, \overline{EP} \notin \mathsf{Fig.eSegments}\}$$

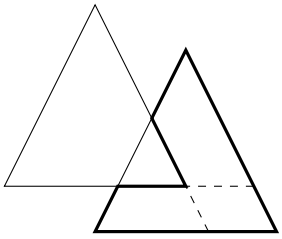


Figure 4.4: Extending Segments for Non-Convex Polygons

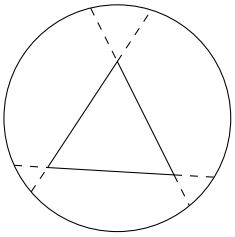


Figure 4.5: Extending Sides of an Orphan Shape

$$\mathsf{Fig.iChords} = \bigcup_{C \in \mathsf{Fig.Circles}} \{\overline{PQ} \mid \forall P, Q \in (\mathsf{Fig.inter})\,.\mathsf{LiesOn}\,(C)\,, \overline{PQ} \notin \mathsf{Fig.eSegments}\}$$

We observe the construction pattern of implied radii and chords in Figure 4.2 where there are three 'open', intersection points resulting in three implied radii and three implied chords.

Construction of radii and chords helps identify a special type of region we refer to as pathological. A *pathological region* is a bounded region that is external to all shapes in a figure. Figure 4.3 defines a pathological shaded area that can be identified using the planar graph technique described in §4.6.1 once implied chords are constructed.

Convexity through Extended Segments. Although most figures in shaded area problems involve convex polygonal shapes, combining those shapes may result in non-convex regions as shown in bold in Figure 4.4.

Our definitions and algorithms rely on the constituent closed objects in a figure be convex. We ensure this convexity from non-convex polygons by extending all applicable sides through the interior (indicated by dashed lines in Figure 4.4). We also extend line segments from orphan shapes (indicated by dashed lines in Figure 4.5). In total, for a figure Fig, we refer to these extended segments as Fig.extSegments and use this set of segments only for atomic region identification (§4.6.1) and not for problem synthesis (§4.7).

4.2.2 Polygon Identification

In simplest terms, a figure is a set of points, segments, and circle-based components. Solving a shaded area problem requires that we identify the set of all polygons in a figure Fig, Fig.Polygons where Fig.Polygons$[i]$ refers to the set of all polygons in Fig with i sides. We describe an algorithm for identifying all polygons (both convex and non-convex) in a given figure Fig. Our analysis assumes input of the set of explicit and implied segments Fig.eSegments $\cup$ Fig.iSegments.

We first identify candidate segments which may be combined into a polygon by eliminating invalid combinations of segments that do not share a vertex or are collinear. Second, we exhaustively construct the set of all triangles in the figure from the set of valid, closed combinations of three segments. Last, we inductively construct polygons of increasing numbers of sides by considering valid sets of segments that do not contain any previously established polygon. That is, for a quadrilateral, we consider all candidate sets of four valid segments as long as no subset of three segments have been used to construct a triangle. We then continue with five segments not containing a triangle nor quadrilateral constructing a pentagon. This method continues inductively to some parameterized upper bound number of sides.

4.3 Shaded Area Problem Formalization

In this section we formally define a shaded area problem and its solution as well as illustrate them via a sample problem and corresponding solution.

Before describing our techniques for solving shaded area problems, we first define some terms related to visual components of a geometry figure.

Definition 25 (Bounded Geometric Object)**.** *A* bounded geometric object *is a simple closed curve (Jordan curve [62]) embedded in the Euclidean plane.*

Definition 26 (Atomic Region)**.** *Given a fixed figure* Fig*, an* atomic region *is a convex, bounded geometric object in* Fig *that has no existing line or arc passing through it.*

In other words, for a fixed figure, the set of all atomic regions is the set of all smallest, closed components of a figure; in Figure 4.16, Fig consists of four atomic regions labeled (1)-(4).

Definition 27 (Region)**.** *A* region *in a figure* Fig *is a non-empty set of atomic regions in* Fig.

We observe for a figure Fig, the set of all regions $\mathsf{Fig.Regions} = \mathcal{P}(\mathsf{Fig.Atoms}) \setminus \emptyset$; in Figure 4.16, Fig consists of $2^4 - 1 = 15$ regions. Thus, computing the regions of a figure requires computing the atomic regions.

Facts. The statement and solution to a shaded area problem requires manipulating two kinds of facts—geometric facts and facts about dimensions.

Definition 28 (Geometric Fact)**.** *A* geometric fact *for a figure* Fig *is a atomic proposition about a figure. We refer to the geometric facts of* Fig *as* $\mathcal{E}(\mathsf{Fig})$*, the explicit facts (as described in §3.2).*

Examples of geometric facts in Figure 4.1 include "OAB is an equilateral triangle" or "$\overline{OM} \cong \overline{ON}$."

Definition 29 (Dimension Fact)**.** *A* dimension fact *for a figure* Fig *is an atomic proposition which relates an object in* Fig *and a numeric constant by equality.*

Definition 30 (Length and Area Facts)**.** *A* length fact *is a dimension fact of the form "*$\mathsf{Length}\,(\ell) = c$*" where* ℓ *refers to a single-dimensional component of* Fig *(segment, angle measurement) and* c *is a numeric constant having a standard unit for length (e.g.* $cm, ft, radians$*, etc.). Similarly, an* area fact *is a dimension fact of the form "*$\mathsf{Area}\,(\alpha) = c$*" where* α *is a two-dimensional component of* Fig *(e.g. circle, atomic region, etc.) and* c *is a numeric constant having a squared unit of measure for area (e.g.* cm^2, ft^2*, etc.).*

An example of a length fact is the measure of an angle is 45^o and an example of an area fact is that the area of circle O is $49\pi\ cm^2$. When context is clear, we will omit units

for readability. For shaded area problems, we need only consider one- and two-dimensional objects and facts; however, extending this notion to $n > 2$ dimensions is possible.

The objective is to compute the area of the shaded portion using geometric reasoning (i.e., logical reasoning using the given facts and the axioms of geometry), area computations of shapes (e.g., computing the area of a circle whose radius is known), and algebraic manipulations (e.g., expressing the area of a region as the sum or difference of two other regions). We may now informally define a shaded area geometry problem.

Definition 31 (Shaded Area Geometry Problem (Informal))**.** *A* shaded area geometry problem $P = \langle \mathsf{Fig}, I, R \rangle$ *consists of (i) a geometric figure* Fig*, (ii) a set of facts* I *we assume about* Fig*, and (iii) the region* R *for which we wish to calculate* $\mathsf{Area}(R)$ *(the 'goal' area fact).*

4.4 Theoretical Foundations for Shaded Area Geometry Reasoning

Following the theoretical foundation described in §3.3, we discuss Euclidean geometry with respect to the shaded area reasoning framework.

Shape Axioms. We add to the universally quantified and existentially quantified axioms with shape axioms (Axm_s): axioms used to calculate the area of a geometric object. That is, we include standard geometric formulae for computing areas lengths and areas in the relations of deduction. If b refers to base length, h refers to height, and d, e, f are the lengths of the sides of a triangle, we have a (non-exhaustive) set of formulae:

- Right Triangles with hypotenuse length f: $d^2 + e^2 = f^2$ (Pythagorean Theorem).
- Triangles: $A = \frac{1}{2} \cdot b \cdot h$.
- Rectangles: $A = b \cdot h$.
- Squares: $A = b^2$.

For trigonometric relations of deduction, we rely upon formulae for triangles. For a triangle with sides a, b, and c and respective opposing angles A, B, and C we have:

Trigonometric Area of Triangles:

$$\mathsf{Triangle}(A,B,C) \Rightarrow \mathsf{Area}\left(\mathsf{Triangle}(A,B,C)\right) = {}^1\!/_2 \cdot a \cdot b \cdot \sin C$$
$$\mathsf{Triangle}(A,B,C) \Rightarrow \mathsf{Area}\left(\mathsf{Triangle}(A,B,C)\right) = {}^1\!/_2 \cdot a \cdot c \cdot \sin B$$
$$\mathsf{Triangle}(A,B,C) \Rightarrow \mathsf{Area}\left(\mathsf{Triangle}(A,B,C)\right) = {}^1\!/_2 \cdot b \cdot c \cdot \sin A$$

Right Triangle Trigonometry:

$$\mathsf{RightTriangle}(A,B,C,\mathsf{Hypotenuse}(c)) \Rightarrow \mathsf{Area}\left(\mathsf{Triangle}(A,B,C)\right) = {}^1\!/_2 \cdot a \cdot b$$
$$\mathsf{RightTriangle}(A,B,C,\mathsf{Hypotenuse}(c)) \Rightarrow A = \arctan\left(\frac{a}{b}\right)$$
$$\mathsf{RightTriangle}(A,B,C,\mathsf{Hypotenuse}(c)) \Rightarrow a = b \cdot \tan A$$
$$\mathsf{RightTriangle}(A,B,C,\mathsf{Hypotenuse}(c)) \Rightarrow b = a \cdot \cot A$$
$$\mathsf{RightTriangle}(A,B,C,\mathsf{Hypotenuse}(c)) \Rightarrow c = \sqrt{a^2 + b^2}$$
$$\mathsf{RightTriangle}(A,B,C,\mathsf{Hypotenuse}(c)) \Rightarrow a = \sqrt{c^2 - b^2}$$
$$\mathsf{RightTriangle}(A,B,C,\mathsf{Hypotenuse}(c)) \Rightarrow b = \sqrt{c^2 - a^2}$$

Figure 4.6: Logical Encoding of Shape Axioms

- Law of Cosines: $a^2 = b^2 + c^2 - 2 \cdot b \cdot c \cos A$.
- Law of Sines: $\frac{\sin A}{a} = \frac{\sin B}{b} = \frac{\sin C}{c}$.
- Area of Triangle: $A = \frac{1}{2} \cdot a \cdot b \cdot \sin C$.
- Right Triangle Trigonometry with hypotenuse c: $\tan A = \frac{a}{b}$, $\sin A = \frac{a}{c}$, and $\cos A = \frac{b}{c}$.

Each shape axiom requires encoding into a logical form. For the trigonometric area of any triangle, we have three distinct deductive encodings shown in Figure 4.6; the remaining encodings are straightforward.

The resulting set of encodings for right triangle trigonometry is extensive since we may solve for any variable in each of the equations. A selection of the encodings are provided in Figure 4.6.

Theory of Area Manipulations. We presume standard notions of areas and their additivity for disjoint components of a figure; likewise, for subtraction of one component completely contained inside another. For example, additivity of Fig in Figure 4.1 allows $\mathsf{Area}\left(\mathsf{Circle}(O, OM)\right) = \mathsf{Area}\left(\mathsf{MajSector}(O,M,N)\right) + \mathsf{Area}\left(\mathsf{MinSector}(O,M,N)\right)$, consequently, $\mathsf{Area}\left(\mathsf{Circle}(O, OM)\right) - \mathsf{Area}\left(\mathsf{MinSector}(O,M,N)\right) = \mathsf{Area}\left(\mathsf{MajSector}(O,M,N)\right)$.

Additivity has ramifications because it implies more properties of a figure, specifically, areas of components of a figure. The definitions are implicit in the theory of a figure Fig as well as the theory of assumed information. For a figure Fig, we call this information the *theory of areas*, $Th\left(\mathsf{Area}^{\mathsf{Fig}}\right)$.

4.4.1 Extending Theories of Figures with Area Computations with a Calculational Logic

Let $\mathcal{C}$ be a logic [22] that extends $\mathcal{L}$ described in §3.3.2 by including descriptions of calculations that may be performed on a geometric figure in order to deduce dimension facts, both lengths and areas.

We define the predicates $\{\mathsf{Known}, \mathsf{Unknown}\}$ to indicate whether a proposition corresponding to a geometry or dimension fact f is an assumption $f \in I$ or is deducible from I: $I \vdash f$. We define $\mathcal{C}$ to include all rules in $\mathcal{L}$ (§3.3) as well as rules such as those stated in Figure 4.7. Rule (1) defines a method by which we can calculate dimension facts from other dimension facts; for example, all of the non-area shape axioms stated in Figure 4.6 such as the Pythagorean theorem. Rule (2) through Rule (4) define methods by which we can deduce dimension facts from problem assumptions and geometric deductions (equality, congruence, and similarity). Rule (5) states that for a shape with the appropriate parameters known according to a geometric area formula, we can compute the area of that shape. Rule (6) through Rule (7) compute areas based on other known areas. We note that computing an area using complementation is accomplished via subtraction.

Theorem 4.4.1 (Completeness of the Calculational Logic)**.** *For a figure* Fig *in a shaded area problem, the area of a region in* Fig *can be calculated using the rules of* $\mathcal{C}$*, assuming the area is computable. That is,* $\mathcal{C}$ *defines a complete logic;*

Proof. Let P be a shaded area problem with figure Fig and assumption facts I. Also let $r \in \mathsf{Fig.Regions}$ be a region.

We pursue completeness as a proof by induction on the depth of a region in a figure. We define a depth function $\mathsf{Depth} : r \to \mathbb{N} \cup \{0\}$, that maps a region to a nat-

$\mathsf{Dim}(X_1, \ldots, X_n)$ refers to a geometric formula by which a dimension parameter Y can be computed from other dimension parameters $X_1, \ldots, X_n$. $\mathsf{Ar}(X_1, \ldots, X_n)$ refers to a geometric area formula for a shape that has parameters $X_1, \ldots, X_n$. $\mathsf{Contains}\,(X, Y)$ is interpreted as an area Y is completely contained within area X. $\mathsf{Disjoint}\,(X, Y)$ means there is no overlap between area of X and area of Y.

1. $$\frac{\mathsf{Known}\,(X_1) \wedge \cdots \wedge \mathsf{Known}\,(X_n) \wedge \mathsf{Dim}(X_1, \ldots, X_n)}{\mathsf{Known}\,(Y)}$$

2. $$\frac{\mathsf{Known}\,(X) \wedge I \vdash X = Y}{\mathsf{Known}\,(Y)}$$

3. $$\frac{\mathsf{Known}\,(X) \wedge I \vdash X \cong Y}{\mathsf{Known}\,(Y)}$$

4. $$\frac{\mathsf{Known}\,(X) \wedge I \vdash X \sim Y}{\mathsf{Known}\,(Y)}$$

5. $$\frac{\mathsf{Known}\,(X_1) \wedge \cdots \wedge \mathsf{Known}\,(X_n) \wedge \mathsf{Ar}(X_1, \ldots, X_n)}{\mathsf{Known}\,(Y)}$$

6. $$\frac{\mathsf{Known}\,(X) \wedge \mathsf{Known}\,(Y) \wedge \mathsf{Disjoint}\,(X, Y)}{\mathsf{Known}\,(X + Y)}$$

7. $$\frac{\mathsf{Known}\,(X) \wedge \mathsf{Known}\,(Y) \wedge \mathsf{Contains}\,(X, Y)}{\mathsf{Known}\,(X - Y)}$$

Figure 4.7: Calculational Logic $\mathcal{C}$ for Computing Dimension Facts

ural number. For $r \in \mathsf{Fig.Regions}$, $\mathsf{Depth}\,(r) = 0$ if r is not contained in any other region in Fig: $\neg\mathsf{Contains}\,(X, r)$ is true for all $X \in \mathsf{Fig.Regions}$. We define the depth of a region r based on the maximum depth of the regions it contains: $\mathsf{Depth}\,(r) = 1 + \max\{\mathsf{Depth}\,(c) \mid \mathsf{Contains}\,(r, c) \wedge c \in \mathsf{Fig.Regions}\}$.

In the base case we consider a set of zero depth regions that define a figure. In this case, $\mathsf{Area}\,(\mathsf{Fig})$ is simply the sum of the constituent regions: $\mathsf{Area}\,(\mathsf{Fig}) = \sum_{r \in \mathsf{Fig.Regions}} \mathsf{Area}\,(r)$. For this reason, we consider a single region $r \in \mathsf{Fig.Regions}$. To perform the deduction

$\mathsf{Ar}(p_1, \ldots, p_n) \vdash \mathsf{Area}(r)$ for $n \geq 1$, we must be able to deduce $\mathsf{Known}(p_1), \ldots, \mathsf{Known}(p_n)$. For all $p \in \{p_1, \ldots, p_n\}$, $\mathsf{Known}(p)$ is true by assumption ($I \models p$) or by deducing dimension facts using Rule (1) through Rule (4). If $\mathsf{Known}(p_1), \ldots, \mathsf{Known}(p_n)$ for all parameters of r, then we may directly compute the area of r via Rule (5): $\mathsf{Ar}(p_1, \ldots, p_n) \vdash \mathsf{Area}(r)$.

We inductively consider each of the area rules in turn: Rule (6) and Rule (7).

Assume that a region r contains $k > 1$ regions, each of known area: for all $c \in \mathcal{C}$, $\mathsf{Known}(\mathsf{Area}(c))$ and $\mathcal{C} = \{c \mid \mathsf{Contains}(r, c) \wedge c \in \mathsf{Fig.Regions}\}$. We note that an arbitrary region is composed of a set of atomic regions. In this case, define $r = \{a_1, \ldots, a_n\}$ for $1 < n \leq k$ and for all $a \in \{a_1, \ldots, a_n\}$, $a \in \mathsf{Fig.Atoms}$ and $\mathsf{Contains}(r, a)$. Using Rule (6) $n - 1$ times, we can additively compute the area of $\mathsf{Area}(r) = \sum_{i=1}^{n} \mathsf{Area}(a_i)$.

With Rule (7), for a region $r \in \mathsf{Fig.Regions}$ assume $\mathsf{Area}(r)$ is known. As with our previous inductive case, we define r as a constituent set of atomic regions $r = \{a_1, \ldots, a_n\}$ for $n > 1$ such that for all $a \in \{a_1, \ldots, a_n\}$, $\mathsf{Contains}(r, a)$. Assume $\mathsf{Area}(r)$ is known, for all $a \in \{a_1, \ldots, a_{n-1}\}$, $\mathsf{Area}(a)$ is known, and $\mathsf{Area}(a_n)$ is unknown. Applying Rule (7) $n - 1$ times with $\mathsf{Area}(r)$ and each $\{a_1, \ldots, a_{n-1}\}$, in turn, results in $\mathsf{Area}(a_n)$. □

Given $\mathcal{C}$ we can formally define the notion of a shape.

Definition 32 (Shape, Root Shape)**.** *For a set of dimension facts D describing a geometric object* Fig *and a shape axiom $A \in \mathsf{Axm}_s$,* $\mathsf{Area}(\mathsf{Fig})$ *is computable using $\mathcal{C}$: strong* $(\mathsf{Fig}) \wedge D \vdash^A \mathsf{Area}(\mathsf{Fig})$*. A shape R is a* root shape *if for all $S \in \mathsf{Fig.Shapes}$, $\neg\mathsf{Contains}(S, R)$ is true.*

A shape is informally a standard geometric objects (square, circle, triangle, etc) in which we can compute the area, if the dimension parameters are known. A root shape is a shape that is not completely contained within any other shape in a given figure.

4.4.2 Synthesis Hypergraph and Problems

As a formal framework, we again use a hypergraph to represent a geometric figure together with the theory of assumptions and theory of areas. Given a geometric figure Fig and a set of geometric axioms Axm, Chapter 3 describes a *logical geometry hypergraph* in

which nodes correspond to $\mathcal{E}(\mathsf{Fig})$ and whose hyperedges are of the form (S, t, A) where $S \vdash^A t$ for $A \in \mathsf{Axm}$. Our solving technique extends a logical geometry hypergraph to additionally track dimension facts in an *analysis hypergraph* as well as include *shape axioms*. We synthesize shaded area problems by exploring this analysis hypergraph.

Definition 33 (Analysis Hypergraph)**.** *Given a figure* Fig*, the* analysis hypergraph *corresponding to* Fig *is* $H_{\mathsf{Fig}}(P, E)$ *where* P *is the set of nodes and* E *is the set of hyperedges. We define the set of nodes in the hypergraph* $P = Th(\mathsf{Fig}) \cup Th(\mathsf{Axm}_x) \cup Th(\mathsf{Axm}_s) \cup Th(K)$ *where* $\mathsf{Fig} = strong(\mathsf{Fig})$*,* Axm_x *is the set of Euclid's axioms,* Axm_s *is the set of shape axioms, and* K *is the student knowledge base. Each hyperedge in* $E \subseteq \bigcup_{i=1}^{|P|} P^i \to P$ *is a function mapping a set of nodes to a single node: if* $Th(\mathsf{Fig}) \cup Th(\mathsf{Axm}_x) \cup Th(\mathsf{Axm}_s) \models (p_1 \wedge \ldots \wedge p_\ell \Rightarrow p)$*, then* $\langle p_1, \ldots, p_\ell \rangle \to p \in E$*.*

Each node in an analysis hypergraph corresponds to a fact about a figure Fig. As described in §4.3 there are two types of facts that are properties of Fig. Nodes in an analysis hypergraph belong to one of two categories $\tau = \{geometric, dimension\}$ We make these distinctions among nodes so that later we may formally define a problem with respect to an analysis hypergraph.

Definition 34 (Geometric and Dimension Nodes)**.** *Let* n *be a node in an analysis hypergraph* H*. If* n *is a propositional formula associated with some* $a \in \mathsf{Axm}_s$*, we say* n *is a* dimension node*. If* n *is a propositional formula associated with some* $a \in \mathsf{Axm}_x$*, we say* n *is a* geometry node*.*

We may further distinguish dimension nodes according to our previous discussions to include *length nodes* and *area nodes*.

Shaded Area Geometry Problems. We informally defined a shaded area problem as a triple in Definition 31. We may now formally define the notion of a shaded area problem as a set of assumptions corresponding to a set of geometric nodes and dimension nodes in an analysis hypergraph and a goal fact corresponding to an area node. The corresponding

The figure at right contains two mutually tangent circles of radii 2cm. The figure defines two distinct goal regions in the figure: α and $\beta = \{\beta_1, \beta_2\}$ as well as three roots shapes (two circles and a square).

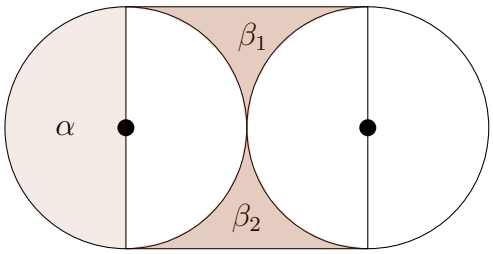


Figure 4.8: Uninteresting Problem Computing $\mathsf{Area}\,(\alpha)$ and Interesting Problem Computing $\mathsf{Area}\,(\beta_1 + \beta_2)$

path from the categorical assumption nodes to the categorical goal node is a solution to the shaded area problem (i.e., deductive proof resulting in computation of the goal area).

Definition 35 (Shaded Area Problem (Formal))**.** *For an analysis hypergraph H_{Fig} corresponding to a figure* Fig*, a* shaded area geometry problem *P is a statement of the form $p_1 \wedge \ldots \wedge p_k \vdash a$ for some $k > 0$ where for all i, p_i is the propositional formula corresponding to the geometric or dimension node n_i of H, a is the propositional formula corresponding to area node g of H, and there exists a path $\mathcal{P}$ from $\langle n_1, \ldots, n_k \rangle$ to g. The path $\mathcal{P}$ is a* solution *to the shaded area geometry problem P. For an area node g, we say that $\mathscr{P}_g$ defines the collection of all paths in hypergraph H with goal area node g; a* valid student solution *is any path in $\mathscr{P}_g$.*

Not all regions in a figure result in meaningful or insightful shaded area problems. We therefore define the concept of *interesting* and *complete* shaded area problems following the paradigm defined in Chapter 3.

Definition 36 (Interesting, Complete Shaded Area Problem)**.** *For a shaded area problem $P = \langle \mathsf{Fig}, I, g \rangle$, if for all $R \in \mathsf{Fig.RootShapes}$, strong $(\mathsf{Fig}) \wedge R \wedge I \vdash \mathsf{Area}\,(g)$ we say P is an* interesting shaded area problem *over* Fig*. A* complete shaded area problem *over* Fig *is an interesting shaded area problem where for all $a \in \mathsf{Fig.Atoms}$, strong $(\mathsf{Fig}) \wedge I \vdash \mathsf{Area}\,(a)$.*

Figure 4.16 is an example of a complete problem since the area of all the atomic regions are computable (hence all regions have computable areas).

For an example of interesting and uninteresting problems, we consider Figure 4.8. In Figure 4.8, α is defined by two roots shapes: the square with one side acting as the diameter of the leftmost circle. For the region defined by α the rightmost circle does not play a role in calculating $\mathsf{Area}\,(\alpha)$ It follows that computing $\mathsf{Area}(\alpha)$ is an *uninteresting* problem. For the region defined by $\beta = \{\beta_1, \beta_2\}$, it is clear that both circles and the square define β. Since all three root shapes are required to compute $\mathsf{Area}(\beta)$, the problem corresponding to computing $\mathsf{Area}(\beta)$ defines an *interesting* problem.

4.5 Figure Synthesis

Synthesis of shaded area problems based on existing figures is, in the end, limiting to the user; new figures along with the corresponding problems are needed. Given a set of shapes, we describe how to synthesize a figure for a shaded area geometry problem and then describe a technique for generating problem assumptions for a given figure.

4.5.1 Figure Synthesis with Templates and Snapping

Our figure synthesis technique is a template-driven approach which defines precisely how one figure is to be composed with another. Figure composition is a challenging proposition with an infinite search space. We overcome this problem using a *template*-based approach with *snapping* in concert with the identification and removal of symmetric figures to limit the search to a finite space and provide meaningful problems.

The central question we attempt to address is: *How can two shapes be combined into a meaningful composition for a shaded area problem?* A quantitative inspection of textbook problems in §4.8 reveals most shaded area figures contain 3 or fewer root shapes as shown in Figure 4.17. A qualitative inspection reveals that typical shaded area problems combine at most 3 shapes (not necessarily root shapes). If we can address the issue of combining two shapes, we can solve the general problem of figure synthesis by repeating our solution to this central question and generate good figures for shaded area problems.

Snapping Points. Not all valid figure compositions are distinct enough for an interesting shaded area problem. That is, minor asymmetric variations in figure compositions may still be deemed similar to a human observer. In order to maximize variation and limit the figure composition search space we use the concept of snapping points. In the context of figure synthesis, snapping points have the same interpretation as they do in any drawing program. A set of *snapping points* for a shape α is the exact set of points for which a shape β may be situated by its snapping points; we note the vertices of a polygon are a subset of its snapping points. The set of snapping points for a particular shape is parameterized. For example, the simplest set of snapping points for shapes may consist of the following:

- all midpoints of segments and midpoints along arcs,
- the center of each circle, and
- quadrantal points of circles (points that lie on the axes in the Cartesian plane at angles 0^o, 90^o, 180^o, and 270^o).

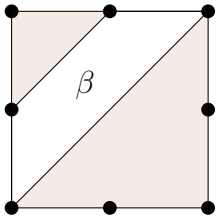
Composition Templates. Let α and β be shapes. The *shaded area subtraction operation* $\alpha - \beta$ refers to β being situated on the interior of α where all outermost vertices (outer snapping points) of β align with snapping points of α; therefore, $\mathsf{Area}\,(\alpha - \beta) = \mathsf{Area}\,(\alpha) - \mathsf{Area}\,(\beta)$. The *shaded area addition operation* $\alpha + \beta$ refers to α being appended to β so that α and β share more than one snapping point and $\alpha \cap \beta = \emptyset$. Since α and β are disjoint, $\mathsf{Area}(\alpha + \beta) = \mathsf{Area}(\alpha) + \mathsf{Area}(\beta)$.

Definition 37 (Shaded Area Composition Template)**.** *A* shaded area figure template *is an expression of the form* $\gamma = \alpha_1 \pm \ldots \pm \alpha_n$ *where* $\alpha_1, \ldots, \alpha_n$ *are shapes and* $\mathsf{Area}\,(\gamma) = \mathsf{Area}\,(\alpha_1) \pm \ldots \pm \mathsf{Area}\,(\alpha_n)$*.*

We remark that neither addition nor subtraction is commutative nor associative.

For both templates $\alpha_1 \pm \alpha_2$, there are clearly an infinite number of satisfiable configurations. Snapping points are used to limit the search space for composition. For shapes α and

Let $\alpha - \beta - \gamma$ be a shaded area template where α is a square, β an isosceles trapezoid, and γ a right triangle. Snapping at midpoints results in one unique asymmetric composition of $\alpha - \beta$ shown at right.



With γ a right triangle, there are 14 such asymmetric compositions for $\alpha - \beta - \gamma$.

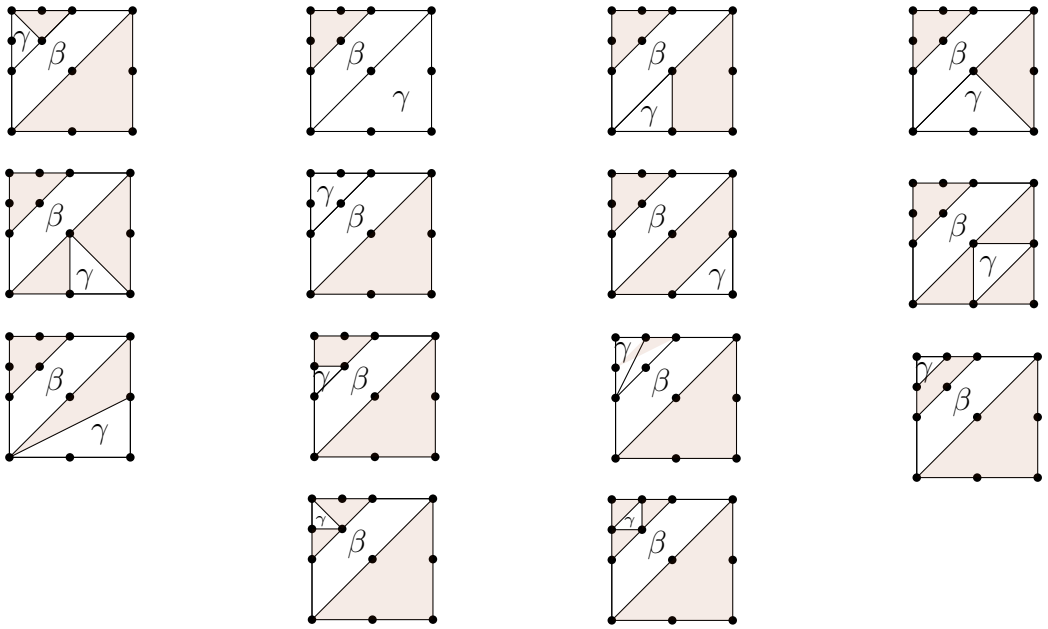


Figure 4.9: Example of Template-Driven Figure Synthesis with $\alpha - \beta - \gamma$

If α is a square and β is an isosceles trapezoid and the depicted set of snapping points are the vertices of the polygons and the midpoints of each side of square, we may construct $\alpha + \beta$ as follows.

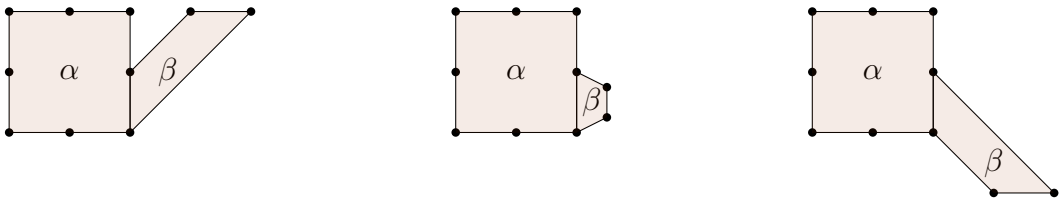


Figure 4.10: Example of $\alpha + \beta$ Figure Composition

β, $\alpha - \beta$ requires all vertices of β align with snapping points of α. Figure 4.9 demonstrates snapping with $\alpha - \beta$ and $\alpha - \beta - \gamma$. Figure 4.10 demonstrates snapping with $\alpha + \beta$.

Depending on the difficulty of the desired problem or to increase the number of possible compositions, we may increase the length of the desired template or granularity of snapping points. One might use more than the midpoints along segments or more than the 16-point unit circle [86] familiar to trigonometry students.

Given a figure generated using our template approach with snapping, we can generate the associated problems using the technique described in §4.7.

4.5.2 Constraint-Based Synthesis of Problem Assumptions From a Figure

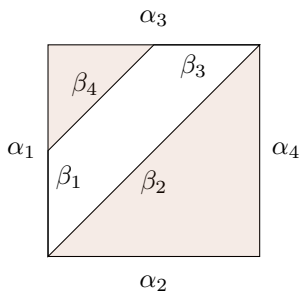
In Chapter 3 we described a technique to identify all minimal sets in a logical geometry hypergraph which in turn become the set of assumptions for a geometry problem. However, since the number of minimal sets increases exponentially with the addition of length-based geometry facts about a figure, we now describe a technique to identify a minimal set of assumptions for a shaded area geometry problem as a heuristic for problem generation from fresh figures.

Defining the shaded area problem $P = \langle \mathsf{Fig}, I, g \rangle$ is accomplished through a constraint-based approach that results in a minimal set of measurements for I. We begin by noting each shape $\alpha \in \mathsf{Fig.Shapes}$ is associated with one or more geometric facts. For example, if α is a rectangle, the corresponding set of geometric facts Known include: each interior angle is a right angle, both sets of opposing sides are congruent and parallel, and the measurements of the sides.

For each shape $\alpha \in \mathsf{Fig.Shapes}$, we use the underlying coordinates to *strengthen* the general shape from an implicit fact to an explicit fact about Fig: $\alpha \in \mathcal{I}(\mathsf{Fig})$ to $\mathit{strong}\,(\alpha) \in \mathcal{E}(\mathsf{Fig})$ where $\mathit{strong} : \mathcal{I}(\mathsf{Fig}) \rightarrow \mathcal{E}(\mathsf{Fig})$ is a function that elevates a shape from a general polygon to a specific polygon (quadrilateral may strengthen to a rectangle).

We now consider how to select which length facts shall be used to compute the area of each shape in $\mathsf{Fig.Shapes}$ for P. Each shape $\alpha \in \mathsf{Fig.Shapes}$ has an associated set of

Let $\alpha - \beta$ be a shaded area template with α a square and β an isosceles trapezoid. A reference construction of $\alpha - \beta$ is depicted where α_i label the sides of α and β_j label the sides of β.



We list a subset of the constraints resulting from $\alpha - \beta$.

- $\alpha_1 \parallel \alpha_4$
- $\alpha_2 \parallel \alpha_3$
- $\beta_1 = \beta_3$
- $\alpha_1 > \beta_1$
- $\beta_2 > \beta_4$
- $\beta_2 \parallel \beta_4$
- $\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4$

There exists an infinite number of isosceles trapezoids that meet the stated constraints. Combining constraint-driven construction of assumptions with snapping limits the cardinality of the set of applicable isosceles trapezoids.

Figure 4.11: Example Set of Constraints Attributed to Composition of Shapes

constraints $K_{strong(\alpha)}$ guided by $strong\,(\alpha) \in \mathcal{E}(\mathsf{Fig})$. For example, there are limited methods to calculate the area of a quadrilateral Q, but recognizing $strong\,(Q)$ as a rectangle means we have the following facts: the opposing sides of $strong\,(Q)$ are congruent and parallel. Hence, for sides of rectangle $strong\,(Q)$, γ_i, $K_{strong(Q)} = \{\gamma_1 = \gamma_3, \gamma_1 \parallel \gamma_3, \gamma_2 = \gamma_4, \gamma_2 \parallel \gamma_4\}$. See Figure 4.11 for an example of constraints attributed to a shaded area template subtraction operation.

To define I as the minimal set of assumptions, we refer to the solution equation E for region g which is a linear combination of areas of regions: $\mathsf{Area}\,(g) = E$. We first construct the set of dependent variables D_g required to calculate $\mathsf{Area}\,(g)$ using E. That is, for each shape $s \in E$, $\mathsf{Area}\,(s)$ is computable if the associated set of parameters P_s are known or calculable using $\mathcal{E}(fig)$ thus satisfying the constraints K_s for shape s. So $D_g = \bigcup_{s \in E} P_s$. We now construct the minimal facts in I iteratively.

1. Randomly select a variable $v \in D_g$.
2. Add v to I: $I := I \cup \{v\}$.

3. Add the shape *strong* (s) associated with v to I: $I := I \cup \{\mathit{strong}\,(s)\}$.
4. Query $\mathcal{E}(fig)$ to identify if the current known set of values I can be used to calculate any other dependent variables in D_g, updating D_g accordingly.

The result is the set I containing a minimal set of assumptions required to calculate $\mathsf{Area}\,(g)$. In the next section we describe how to solve a shaded area problem through reachability in an analysis hypergraph.

4.6 Solving Shaded Area Problems

In this section we describe *GeoShader*, our tool for solving shaded area problems. For a shaded area problem $P = \langle \mathsf{Fig}, I, g \rangle$, the input is Fig which has been analyzed according to §4.2. We then continue processing Fig to identify the set of atomic regions. Last, we construct an analysis hypergraph relating the geometric and dimension facts of Fig and traverse the hypergraph to identify a solution to P.

4.6.1 Atomic Region Identification

In this subsection we describe the challenges associated with converting a figure to a planar graph using a disambiguation process, but first we describe how to compute facets of a a planar graph (which correspond to the atomic regions of a figure).

Facet Identification for a Planar Graph. Atomic region identification is accomplished by identifying the the smallest, bounded regions of a planar graph (commonly referred to as facets [35]).

Definition 38 (Planar Graph)**.** *A* planar graph $G_p\,(N_p, E_p)$ *is an undirected graph embedded in the Euclidean plane where* N_p *is a set of points in the Euclidean plane. Each planar edge* $e_p = (s, t) \in E_p$ *is defined as a segment with endpoints* $s, t \in N_p$*.*

Facet identification for a planar graph embedded in the plane is a well-known problem and is described in [29] and explained in detail in [34]. For clarity, we present pseudocode for facet identification in Algorithm 4.1. Given a planar graph G_p (Line 1), Algorithm 4.1 identifies and returns the corresponding facets (simple cycles in G_p on Line 2).

Algorithm 4.1 Facet Identification in a Planar Graph

```
1: procedure FacetIdentification(G_p (N_p, E_p): Planar Graph)
2:     𝒞 ← ∅: Cycles
3:     while N_p ≠ ∅ do
4:         s ← N_p.least()                          ▷ Least Lexicographic Point
5:         C ← {s}: Cycle
6:         p ← s
7:         c ← CounterClockNeighbor(G_p, s)
8:         switch |Adjacent(s)| do
9:             case 0
10:                N_p ← N_p \ {s}
11:            case 1
12:                N_p ← N_p \ {s}
13:                E_p ← E_p \ {(s, c)}
14:            case > 2
15:                while c ≠ s do                   ▷ Extract a Cycle
16:                    C ← C ∪ {c}
17:                    n ← CounterClockNeighbor(G_p, p, c)
18:                    p ← c
19:                    c ← n
20:                end while
21:                𝒞 ← 𝒞 ∪ {C}
22:                E_p ← E_p \ {(s, c)}
23:    end while
24:    return 𝒞
25: end procedure
```

Identification of simple cycles continues until we have exhausted the set of points (Line 3). In the (outer) loop, we begin cycle identification with the lexicographically 'least' point (Line 4). We are most interested in simple cycles in G_p (Line 14), but as we modify G_p, orphaned points (Line 9) or points with no return edges (Line 11) may arise so we check the number of adjacent points from s by calling Adjacent(s). Cycle identification involves greedy point selection in a counterclockwise manner (Line 15 through Line 20). The first call to CounterClockNeighbor(G_p, s) chooses the point that creates the smallest angle with the downward reference vector from s. The subsequent call to CounterClockNeighbor(G_p, p, c) chooses the point creating the smallest counterclockwise angle measured with respect to the reference vector $\overrightarrow{pc}$. Once a cycle is acquired,

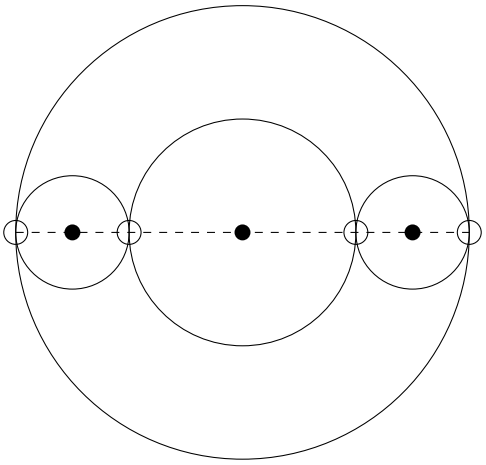


Figure 4.12: Preprocessing a Figure

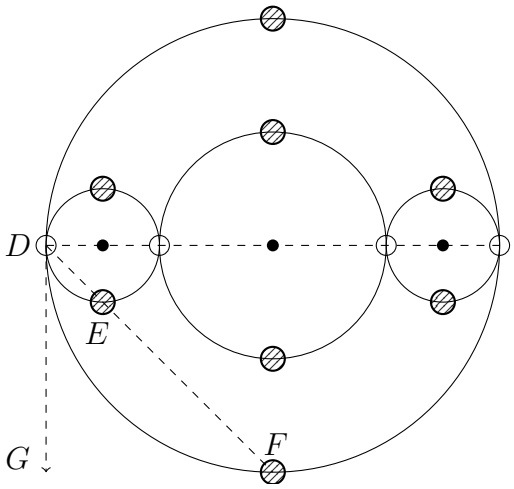


Figure 4.13: A Failed Disambiguation Scheme for a Figure

we extract the facet (corresponding to an atomic region in Fig) and remove the first edge (Line 22) from G_p so the first edge is never taken again. We repeat this process identifying all facets of G_p.

Ambiguity of the Planar Graph Corresponding to a Figure. The atomic regions in a figure Fig corresponds to the facets in a corresponding planar graph $G_p\,(N_p, E_p)$ using Algorithm 4.1 assuming that G_p completely defines Fig. For figure Fig illustrated in Figure 4.12, the set of points Fig.DefinePts are the dark, 'filled' points and Fig.inter are the 'open' points. It is clear that if E_p = Fig.iRadii and N_p = Fig.DefinePts $\cup$ Fig.inter in G_p, no facets are identifiable in the corresponding planar graph. In this case, the planar graph does not completely define Fig; ambiguities arise from arcs and circles.

We require more information to completely define Fig with a corresponding planar graph. If we define a set of disambiguating points D as the larger points that are 'shaded' with lines in Figure 4.13 an ambiguity persists when defining N_p = Fig.DefinePts$\cup$Fig.inter$\cup$ D. For Fig depicted in Figure 4.13, Algorithm 4.1 begins facet identification from point D. Now, we greedily seek the next counterclockwise point resulting from the reference vector $\overrightarrow{DG}$. The next point in a counterclockwise traversal should be F; however, points D, E,

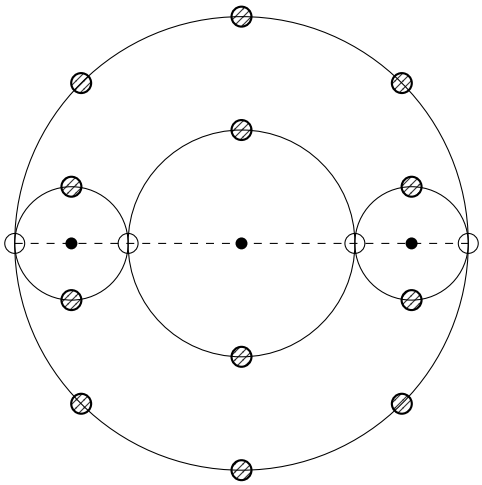


Figure 4.14: Minimal Disambiguating Set of a Figure

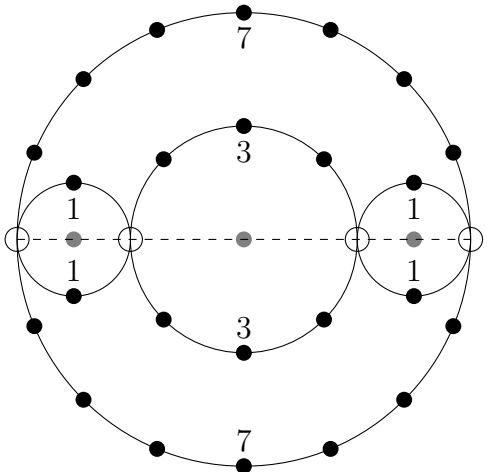


Figure 4.15: Automated Disambiguation Set of a Figure

and F are collinear and thus $\angle FDG \cong \angle EDG$. Without the inclusion of another point along $\overset{\frown}{\mathrm{DF}}$, the choice of the next edge from D is ambiguous: we cannot distinguish the edge D to E from the edge from D to F. Adding a single point to all arcs is not sufficient to define a figure through a planar graph; we must be able to distinguish between segments and arcs in all situations.

Resolving the Ambiguity of a Planar Graph Corresponding to a Figure. To resolve the ambiguity that arises when constructing a planar graph from a figure Fig, we need to define a new set of constructed, disambiguating points we call Fig.Disambiguating along each arc. Figure 4.14 demonstrates one successful addition of (dark) points for the planar graph to uniquely determine the figure; this is the minimal set of points required to define Fig as a planar graph. Our approach to resolving these ambiguities is based on the number and size of circles in Fig. Sorting the circles by radius length from least to greatest, we add an exponentially increasing number of points (1, 3, 7, etc.) at constant intervals along all arcs. Our automated approach for disambiguation is shown in Figure 4.15 where Fig.inter are 'open' points and Fig.Disambiguating are the dark points; the numeric values in Figure 4.15 refer to the number of disambiguating points added along each arc (between intersection points).

Our approach does not minimize the number of constructed points required to resolve ambiguity, but successfully resolves such ambiguities for standard shaded area problems. We informally reason that, in the case of Figure 4.13, if the respective angles formed by tangent $\overrightarrow{DG}$ and the rays $\overrightarrow{DE}$ and $\overrightarrow{DF}$ are equal in measure, the arcs are equal in measure. By subdividing the smaller arc into 2^n equal subarcs by inserting $2^n - 1$ constructed points and subdividing the larger arc into 2^m equal subarcs by inserting $2^m - 1$ constructed points (where $m > n$) ensures that all subarcs in the larger circle measure less than all subarcs in the smaller circle. Hence, circles with distinct radii lengths will have distinct counter-clockwise angles with respect to $\overrightarrow{DG}$. If it is the case that the respective angles formed by tangent circles does not result in arcs that are equal in measure, we modify our argument to account for the ratio between the two measures. This technique removes ambiguity in general for any tangent situation for two circles intersecting by defining $\mathsf{Fig.Disambiguating}$ for a figure Fig.

Identification of Atomic Regions for a Figure. To compute $\mathsf{Fig.Atoms}$ for a figure Fig, we construct a planar graph $G_p\,(N_p, E_p)$ in which $N_p = \mathsf{Fig.DefinePts} \cup \mathsf{Fig.inter} \cup \mathsf{Fig.Disambiguating}$ and edges appropriately connect elements of N_p using the set of arcs and both implied and explicit segments: $\mathsf{Fig.Arcs} \cup \mathsf{Fig.eSegments} \cup \mathsf{Fig.iSegments}$. The atomic regions $\mathsf{Fig.Atoms}$ correspond to the facets of G_p computed using $\textsc{FacetIdentification}(G_p)$ as defined in Algorithm 4.1. As an example set of atomic regions, for figure Fig in Figure 4.15 $\mathsf{Fig.Atoms}$ contains six semicircles and two symmetric atomic regions inside the outer circle, but outside the three smaller circles.

4.6.2 Constructing the Analysis Hypergraph

For a figure Fig, the corresponding analysis hypergraph H_{Fig} is composed of geometric facts ($H_{\mathsf{Fig}}.\mathcal{E}(\mathsf{Fig})$), length ($H_{\mathsf{Fig}}.\mathsf{Length}$), and area facts ($H_{\mathsf{Fig}}.\mathsf{Area}$); we note the set of dimension facts in an analysis hypergraph are given by $H_{\mathsf{Fig}}.\mathsf{Dimension} = H_{\mathsf{Fig}}.\mathsf{Length} \cup H_{\mathsf{Fig}}.\mathsf{Area}$. Since the analysis hypergraph is an extension of the logical hypergraph in Chapter 3, we construct $H_{\mathsf{Fig}}.\mathcal{E}(\mathsf{Fig})$ using the technique described in §3.4.1 for geometric

facts. Since a figure Fig is immutable, each single-dimensional measurement (angle measure, segment length, etc.) is added to the analysis hypergraph as an element of $H_{\mathsf{Fig}}.\mathsf{Length}$; for example, $OM = 7$ and $OA = 12$ in Figure 4.1 are length facts. The set of area facts $H_{\mathsf{Fig}}.\mathsf{Area}$ are computed using two techniques: (1) the first relates geometric facts and length facts and (2) the second relates area facts to other area facts using addition or subtraction.

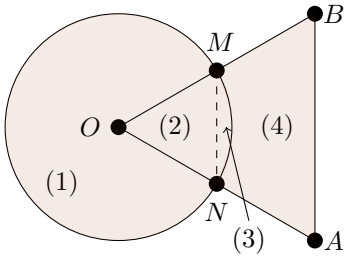
We first compute area facts using shape axioms for figure Fig. For each shape $s \in \mathsf{Fig.Shapes}$, we add a corresponding hyperedge to H_{Fig} for all $D \subset H_{\mathsf{Fig}}.\mathsf{Dimension}$, for all $A \in \mathsf{Axm}_s$, if $\mathit{strong}\,(s) \wedge D \vdash^{A} \mathsf{Area}\,(s)$. We note in Figure 4.16 that $\triangle OAB$ corresponds to region $\{(2), (3), (4)\}$ and in Figure 4.1 we use the length facts $OA = 12$ and $\angle BOA$ measures 60^o to deduce $\mathsf{Area}(\triangle OAB) = 36\sqrt{3}$ by way of the area formula for an equilateral triangle.

Deducing an area fact from two area facts by means of addition or subtraction of the respective areas is a simple process, but is computationally expensive. This is due to the fact that the number of facets of a planar graph is linear in the size of the graph and the number of regions, corresponding to sets of facets, is thus exponential. We therefore do not construct the entire analysis hypergraph for a given figure, but can limit construction of nodes to the set of assumptions in the problem, if available. We use the following algorithm as a heuristic to avoid area facts that are not computable with the problem parameters. We deduce an area fact from sets of area facts using an algorithm composed of two parts. (1) Organize the shapes into a hierarchy, computing areas of regions traversing down the hierarchy. (2) Use a fixed-point approach to compute areas of regions that are unions or differences of two regions by respectively adding or subtracting known areas.

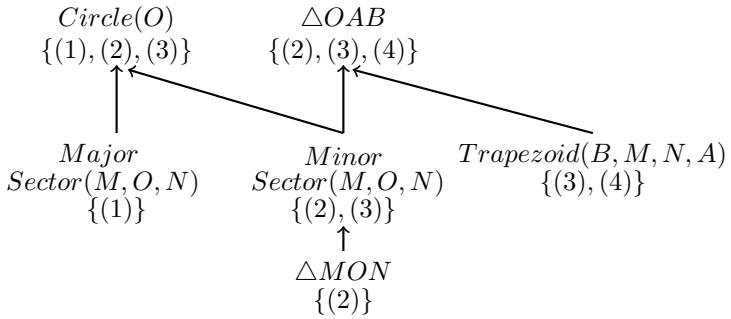
Deducing Area Facts from Area Facts Using a Shape Hierarchy. Instead of exhaustively exploring all possible relationships among subsets of atomic regions, we use a hierarchy of shapes as a heuristic. This hierarchy mimics the set of shapes a student may identify and employ in their solving.

For a figure Fig, we organize shapes as a directed acyclic graph called the *shape hierarchy*. The roots of the shape hierarchy are Fig.RootShapes. We construct the shape hierarchy

We reconsider the problem in Figure 4.1 with annotated figure Fig at right.
For *atomic region identification* we construct chord $\overline{MN}$ resulting in four atomic regions labeled (1)-(4) (thus $2^4 - 1 = 15$ regions).



We have the corresponding *shape hierarchy* for Fig noting that Fig consists of a circle, two sectors, a trapezoid, and two triangles.



Computability of Shape Areas. Most of the shapes have computable areas using standard shape axioms. Thus the areas of the corresponding regions are computable as well.

$\mathsf{Area}(\{(1),(2),(3)\}) = 49\pi$
$\mathsf{Area}(\{(1)\}) = \frac{245}{6}\pi$
$\mathsf{Area}(\{(2),(3),(4)\}) = 36\sqrt{3}$
$\mathsf{Area}(\{(2),(3)\}) = \frac{49}{6}\pi$
$\mathsf{Area}(\{(2)\}) = \frac{49}{4}\sqrt{3}$

Hierarchical Subtraction. We demonstrate a few of the hierarchical subtraction operations that result in more computable region areas.

$\{(1),(3)\} = \{(1),(2),(3)\} - \{(2)\}$
$\{(3)\} = \{(2),(3)\} - \{(2)\}$
$\{(4)\} = \{(2),(3),(4)\} - \{(2),(3)\}$
$\{(3),(4)\} = \{(2),(3),(4)\} - \{(2)\}$

Fixed-Point Combining. Shape subtraction is not satisfactory to solve this problem since it seeks the area of the entire figure. We combine all existent region expressions to acquire the shortest solution found by our tool *GeoShader*.

$$\mathsf{Area}(\{(1),(2),(3),(4)\}) = \mathsf{Area}(\{(1)\}) + \mathsf{Area}(\{(2),(3),(4)\}) = \frac{245}{6}\pi + 36\sqrt{3}$$

Figure 4.16: Solving the Shaded Area Problem of Figure 4.1

by noting that the children of a node are shapes that are fully contained in the shape corresponding to their parent node in the hierarchy. In Figure 4.16, $\mathsf{MinSector}(M,O,N)$ is directly contained within both $\mathsf{Circle}(O,OM)$ and $\mathsf{Triangle}(O,A,B)$ so there exists directed edges in the associated shape hierarchy from $\mathsf{MinSector}(M,O,N)$ to both shapes.

Given a shape hierarchy, we mimic how a student may approach handling area calculations by taking a series of differences between a node and all of its descendants. That is, for each $s \in \mathsf{Fig.Shapes}$, for each $c \in \mathsf{Fig.Shapes}$ $(c \neq s)$, if $\mathsf{Contains}(c, s) \wedge \mathsf{Known}(\mathsf{Area}(c)) \wedge \mathsf{Known}(\mathsf{Area}(s)) \vdash \mathsf{Known}(\mathsf{Area}(c - s))$ we add a corresponding hyperedge to the analysis hypergraph. For example, in Figure 4.16, we can compute the area of region $\{(1), (3)\}$ by taking the difference between $\mathsf{Circle}(O, OM)$ and $\mathsf{Triangle}(M, O, N)$: $\mathsf{Area}(\{(1), (3)\}) = \mathsf{Area}(\mathsf{Circle}(O, OM)) - \mathsf{Area}(\mathsf{Triangle}(M, O, N)) = 49\pi - \frac{49\sqrt{3}}{4}$. Similarly, we may compute the area of region $\{(3), (4)\}$ which defines $\mathsf{Trapezoid}(B, M, N, A)$ as $\mathsf{Area}(\mathsf{Triangle}(O, A, B)) - \mathsf{Area}(\mathsf{Triangle}(M, O, N)) = \mathsf{Area}(\{(3), (4)\}) = 36\sqrt{3} - \frac{49\sqrt{3}}{4}$.

Deducing Area Facts from Area Facts With Fixed-Point Combining. Thus far we have computed facts for areas of regions as (1) directly from shape axioms and (2) subtraction of areas with the shape hierarchy; we refer to these facts as $\mathcal{K}$. Our last step in constructing area facts uses a fixed-point approach to computing the areas of additional regions. That is, for each $a_1, a_2 \in \mathcal{K}$, if $\mathsf{Contains}(a_1, a_2) \wedge \mathsf{Known}(\mathsf{Area}(a_1)) \wedge \mathsf{Known}(\mathsf{Area}(a_2)) \vdash \mathsf{Known}(\mathsf{Area}(a_1 - a_2))$ we add a corresponding hyperedge to the analysis hypergraph. Similarly, we add a corresponding hyperedge to the analysis hypergraph if $\mathsf{Disjoint}(a_1, a_2) \wedge \mathsf{Known}(\mathsf{Area}(a_1)) \wedge \mathsf{Known}(\mathsf{Area}(a_2)) \vdash \mathsf{Known}(\mathsf{Area}(a_1 + a_2))$.

Finding the area of a goal region in some shaded area problems does not require this step; however, in the case of Figure 4.16 solving the problem is impossible without this algebraic fixed-point process. In Figure 4.16 we know $\mathsf{Area}(\mathsf{MajSector}(M, O, N)) = \frac{245}{6}\pi$ and $\mathsf{Area}(\triangle OAB) = 36\sqrt{3}$ with respective regions $\{(1)\}$ and $\{(2), (3), (4)\}$. Taking the union of the two regions results in the solution to the problem $\mathsf{Area}(\mathsf{Fig}) = \mathsf{Area}(\{(1), (2), (3), (4)\}) = \frac{245}{6}\pi + 36\sqrt{3}$. We note that this algebraic combining is how we initially solved the problem in Figure 4.1.

4.6.3 Finding a Path in the Hypergraph

As noted in Definition 35, a solution to a shaded area problem is a path in the corresponding analysis hypergraph. Our goal is to identify such a path for some shaded area

problem $P = \langle \mathsf{Fig}, I, g \rangle$. Identifying a solution to problem P consists of two distinct steps. The first step takes Fig and uses the process described in §4.6.2 to construct the corresponding analysis hypergraph, H_{Fig}. The second step is to identify a path from the nodes corresponding to I and the node corresponding to the area fact $\mathsf{Area}\,(g)$ in H_{Fig}. Identifying the solution to P corresponds to reachability from the node corresponding to $\mathsf{Area}\,(g)$ to the nodes corresponding to I in ${H_{\mathsf{Fig}}}^T$ as described in §2.3. The resultant solution is the path representing the solution to problem P. See Figure 4.1 for a solution constructed by *GeoShader* for the stated problem.

4.7 Problem Generation

A student studying for an exam or a teacher attempting to construct novel questions are limited by their resources and ingenuity. With that in mind, we present an algorithm for synthesizing interesting shaded area problems based on an existing figure either acquired from a slate or from figure synthesis described in §4.5.

The problem synthesis algorithm is quite simple because it relies upon previous results described in Chapter 3. Given a figure Fig, we construct the analysis hypergraph as described in §4.6 where the set of nodes are the geometric facts and area facts for each region in Fig and hyperedges correspond to geometric deduction.

To acquire the set of shaded area problems based on Fig we use the *GenProblem* algorithm described in Algorithm 3.2 in §3.4 where the goal is a singleton area fact node labeled $\mathsf{Area}(g) = c$ for some constant c where g is a region in Fig. We restrict our problem synthesis to interesting shaded area problems based on whether a problem with goal region g meets the criteria for an interesting problem.

4.8 Experimental Results

Evaluation Criteria. We first describe our benchmark set of problems and characteristics of the corresponding figure. Second, we evaluate our solution technique with respect to time required to construct the corresponding hypergraph and identify the solution path.

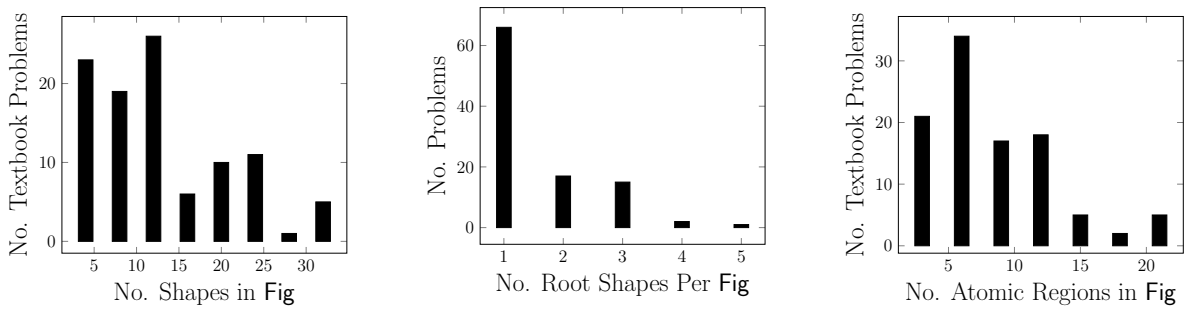


Figure 4.17: Characteristics of Textbook Problems

Last, we correlate structural characteristics of a solution with respect to the time taken to generate that solution.

We ran our solution generation algorithm on a laptop with Intel Core i5-2520M CPU at 2.5GHz with 8 GB RAM on 64-bit Windows 7 operating system.

Benchmark Problem Set. We ran our solution generation algorithm on a set of 102 figures taken from standard mathematics textbooks and workbooks from the United States [51, 47, 56, 16, 23] as well as released exams from the Indian Class X exam [21]. We used a uniform set of geometric area formulas and geometric axioms for all of our experiments: tangent relationships to circles, quadrilaterals, congruent triangles, etc.

In the set of 102 figures from textbook problems we observe a figure in a shaded area problem has mean (and standard deviation) 11.5 (7.8) shapes 1.56 (0.88) root shapes, and 7.3 (4.6) atomic regions. The number of shapes, root shapes, and atomic regions per problem result in right-skewed distributions as shown in Figure 4.17.

Problem Solving vs. Time. As described in §4.6, solving a shaded area problem requires computing the atomic regions in the figure (§4.6.1), construct the logical hypergraph, constructing the analysis hypergraph (§4.6.2) and path identification of the solution (§4.6.3); Figure 4.18 shows the time required for each of the three phases. We note a mean (and standard deviation) of 2.79 (2.53) seconds for atomic region identification, 7.29 (12.10) seconds for deduction engine construction, 3.33 (7.91) seconds for area fact deduction and computing the solution, and overall time 13.4 (17.24) seconds.

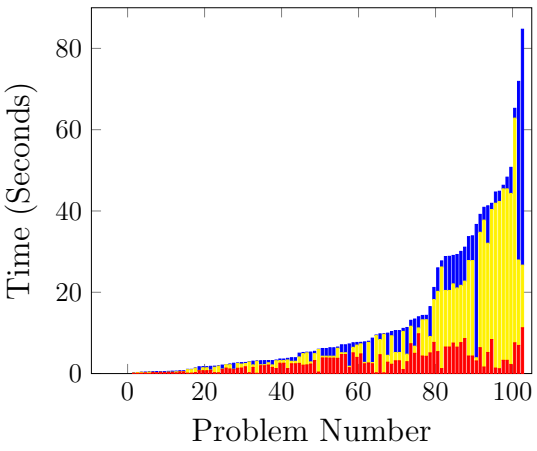


Figure 4.18: Sorted Times for Finding the Solution to a Shaded Area Problem: Atomic Region Identification (red), Deduction Engine (yellow), and Computing the Solution and its Features (blue)

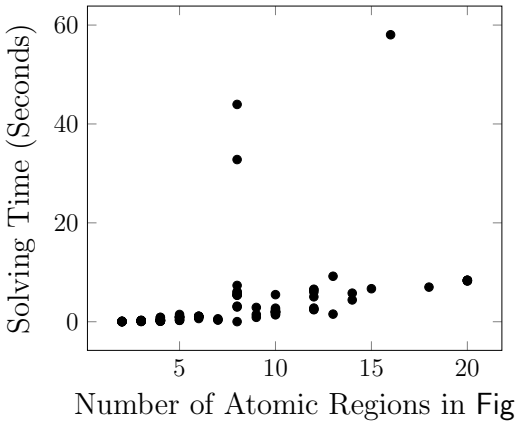


Figure 4.19: Number of Atomic Regions Compared to Solving Time

We note that in constructing the nodes and edges in the analysis hypergraph, which avoids eager consideration of the exponential number of regions, is thus well-motivated since the number of atomic regions can often be too large. In Figure 4.19 we see a positive correlation ($r^2 = 0.599$) for an exponential regression when we consider the number of atomic regions compared to the length of time for the last phase of the solution process: area facts and deductions as well as compute the solution.

Solution Characteristics. As defined in §4.6, the solution to a shaded area geometry problem is a DAG and therefore has several quantitative features. For example, the *depth*

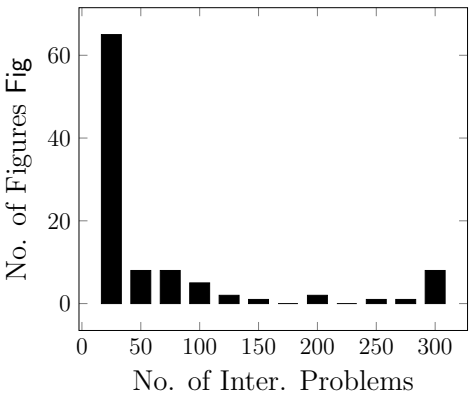


Figure 4.20: Histogram of Generated Interesting Problems by Fig

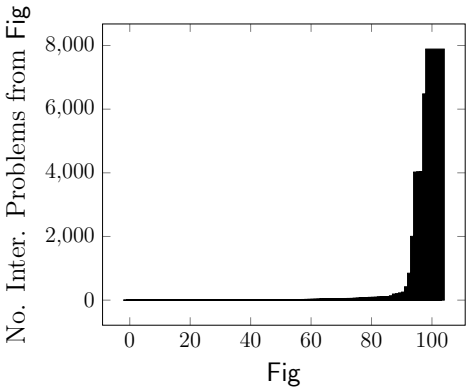


Figure 4.21: (Sorted) Number of Generated Interesting Problems Per Fig

Table 4.1: Synthesis with Existing Figures: Mean (Standard Deviation)

	Figures	Atomic Regions	Generated Interesting Problems
Without Circles	25	4.44 (2.66)	10.28 (15.61)
With Circles	77	7.65 (4.16)	336.7 (1188.1)
Composite	102	6.86 (4.11)	256.7 (1040.3)

of a solution is the longest path from the assumptions to the area in the solution, *width* is maximal number of nodes in a level, and number of *deduction steps* corresponding directly to the number of hyperedges in the solution. With our solutions to the 102 shaded area problems, we see a mean (and standard deviation) for depth 7.0 (2.5), width 6.8 (3.8), and number of deductions 11.9 (8.0). For the solutions, we observe mean 13.1 (8.2) geometry facts and 2.1 (0.9) area facts.

Effectiveness of Problem Synthesis on Existing Figures. Using the 102 textbook figures as a basis of analysis, we show the effectiveness of our problem synthesis algorithm with a mean of 256.7 (1040.3) interesting problems per figure Fig as stated in Table 4.1. Figure 4.20 shows that most figures result in a small number (less than 25) of generated problems while some figures result in thousands of problems (as shown in Figure 4.21).

We consider the relationship between the number of solvable regions compared to the number of generated problems with respect to Table 4.1. Solving shaded area problems according to the algorithms described in §4.6 requires construction of radii and chords

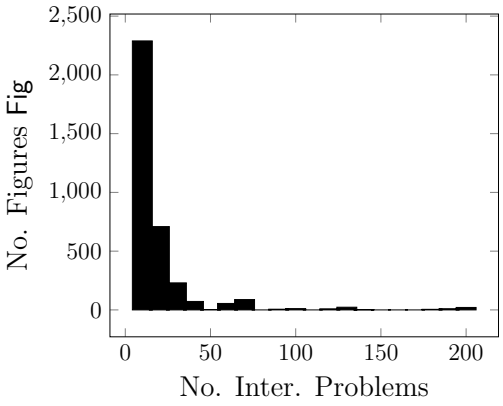


Figure 4.22: Histogram of Interesting Problems Generated Per Synthesized Figure Fig

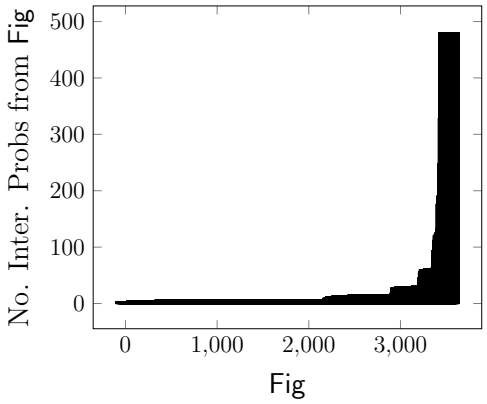


Figure 4.23: (Sorted) Generated Interesting Problems Per Synthesized Figure Fig

Table 4.2: Figure and Problem Synthesis for $\alpha, \beta, \gamma \in \{\text{Square}, \text{Rectangle}, \text{Right Triangle}\}$: Mean (Standard Deviation)

Template	Figures	Atomic Regions	Generated Interesting Problems
$\alpha - \beta$	52	2.63 (0.93)	6.92 (7.42)
$\alpha + \beta$	67	2.50 (0.5)	3 (0.0)
	119	2.56 (0.72)	4.71 (5.25)
$\alpha - (\beta - \gamma)$	505	5.56 (2.67)	37.2 (73.40)
$(\alpha - \beta) - \gamma$	705	4.91 (2.18)	26.0 (31.24)
$(\alpha - \beta) + \gamma$	623	4.62 (1.35)	14.4 (19.99)
$\alpha + \beta + \gamma$	994	4.78 (0.83)	6.49 (1.83)
$(\alpha + \beta) - \gamma$	587	4.44 (1.08)	8.79 (5.88)
	3414	4.83 (1.70)	16.65 (34.63)
Overall	3533	4.76 (1.73)	16.51 (34.12)

when the figure includes a circle; construction of segments results in more atomic regions in the figure. In Table 4.1 we see 77 of the 102 textbook problems involve circles. Those 77 problems involving circles have a significant increase in the number of atomic regions (7.65) compared to the 25 problems that do not involve circles (4.44 atomic regions). The number of atomic regions thus influences the number of interesting problems; 10.28 problems without circles compared to 336.7 with circles, a significant disparity.

Effectiveness of Problem Synthesis through Figure Synthesis. We evaluate the figure synthesis technique described in §4.5 by limiting the search space with the selection of shapes, snapping points, and templates. In this analysis we considered the set of shapes

which included squares, rectangles, and right triangles. We used the midpoints of segments as snapping points. We consider each of the templates listed in Table 4.2 with the set of shapes $\{\mathsf{Square}, \mathsf{Rectangle}, \mathsf{RightTriangle}\}$. For each template, we generated a minimal set of assumptions using the technique described in §4.5.2. Under these conditions, we generated 3533 figures with corresponding mean 16.5 (34.1) interesting problems.

We compare our figure synthesis procedure to existing figures. Visually, we see a similar shape in the distribution in Figure 4.22 for figure synthesis compared to Figure 4.20 with existing figures; similarly Figure 4.23 is comparable to Figure 4.21. Numerically, existing figures without circles have a mean of 10.28 interesting problems compared to 16.51 for synthesized figures without circles. We attribute the slight difference to the number of atomic regions in the figures: mean atomic regions is slightly larger for synthesized figures without circles (4.76) compared to existing figures without circles (4.44).

Last, we consider the number of atomic regions as a feature of a figure with circles. According to Table 4.1, a mean of 7.65 atomic regions results in 336.7 interesting problems. For $\beta = \{\mathsf{Isosceles\ Trapezoid}, \mathsf{Right\ Triangle}, \mathsf{Rectangle}, \mathsf{Equilateral\ Triangle}\}$, the template $\mathsf{Circle} - \beta$ resulted in figures with 11.0 (4.24) atomic regions. We observe figure synthesis results in more atomic regions compared to existing figures. We attribute this difference due to our definition of subtraction with templates requiring one shape to be completely contained within another whereas existing figures may orient shapes such that one shape intersects another, but either shape is not completely contained within the other.

4.9 Related Work in Geometry Problem and Solution Synthesis

We discuss work related to automated problem and solution generation for high school geometry problems as well as template-based problem generation with respect to geometry proof problem synthesis with *GeoTutor* and shaded area problem solving and synthesis with *GeoShader*.

4.9.1 Automated Tutoring Systems

Existing automated tutoring systems provide varied levels of personalized feedback to students. Wolfram Alpha [1] provides step-by-step solutions and hints for computational domains. AutoTutor [2] is an interactive dialogue-based tutoring system for physics and computer literacy that provides feedback to students of all ability levels. However, these systems do not automatically synthesize analogous exercises to provide personalized practice to a student having difficulties in particular areas or types of exercises. Individualized, but analogous assignments can mitigate cheating while maintaining fairness. None of the existent systems cover difficult topics like Geometry. Unlike *GeoTutor*, these systems provide problems from a predefined set that are slightly modified versions of those scoured from a plethora of textbooks or in the case of Wolfram Alpha, generated from discrete, algebraic problem templates. *GeoTutor* and *GeoShader* can synthesize problems beyond those available in textbooks; the student is free to generate their own problems by creating their own figures and associated assumptions.

4.9.2 Technology for Geometry Education in Proof Synthesis

Automated geometry theorem proving (consisting of several techniques such as Wu's method [82], Grobner basis method [52], and angle method [24]) is one of the most successful areas of automated reasoning. Traditional automated geometry theorem proving systems tend to produce arbitrary proofs in the underlying logical domain that may not be readable and may be beyond the vocabulary taught in the class. Tutoring oriented systems such as Geometry Expert [39] and Geometry Explorer [87] allow students to create geometry constructions and use interactive provers to check and prove properties of those constructions. [44, 49] even present techniques for automatically synthesizing geometry constructions given logical constraints that relate the various objects in the construction.

The *GeoTutor* system can be used to solve those proof problems that do not require construction of any new object in the given geometric figure. It uses a relatively simple methodology of hypergraph reachability to check whether the goal can be reached from the

assumptions. The novelty of our system lies in the hypergraph construction and associated algorithms over it that enables generation of various interesting problems.

4.9.3 Technology for Geometry Education in Shaded Area Synthesis

Our work with *GeoTutor* first formalized the notion of implicit and explicit atomic geometry facts in a given geometry figure as well as rules for reasoning over those facts. With *GeoShader* we extend that formalism to deal with a richer class of facts involving area facts and rules that relate these facts with each other and also with atomic geometry facts. More significantly, we address the novel challenge of parsing a given coordinate-based geometry image into implicit facts related to both atomic properties and area properties. We also present an approach for synthesizing new geometry figures that can be used to construct new problems unlike past work [7] that is restricted to considering only existing figures. [63] also addressed solution generation for a wide range of mathematics problems including analytic geometry based solely on a textual description. We use a pixel-based approach and reason about existing figures in our solution generation.

Recently, [74] describe a technique of diagrammatic understanding by extracting implicit atomic geometry facts from a figure using vision-based techniques. We present a distinct technique to address a more involved problem of also extracting area geometry facts.

4.9.4 Automatic Problem Generation

[77] describes a template-based problem generation technology for generating problems where the input problem defines the structural template for a given algebraic identity proof problem. Our figure synthesis technique does not require an input problem as stimulus, but allows the user to specify a general set of interactions among the figures through the template which influences the structural nature of the resultant figure.

Problem generation technologies exist for a *procedural domain* [10] in which problems are generated for various paths a student is required to learn in a given procedure. In

contrast, we address problem generation for a *conceptual domain* where there is no single step-by-step decision procedure that the student can use to solve a problem. The conceptual domain of problems requires creativity in solving such as induction, deduction, or pattern matching.

[7] describes a problem generation technique that represents all possible applications of the various axioms and traverses that graph to simultaneously construct new interesting problems and their solutions. This is similar to the technique in [5] for natural deduction problems which constructs a Universal proof hypergraph of all possible inference rule applications and traverses this graph to generate problems with certain features. Our technique for area reasoning problems is similar in that we use hypergraph construction for solution generation and interesting problem generation.

Chapter 5
Molecular Synthesis

In this chapter we discuss background in molecular synthesis and significance of the antibiotic resistance problem, introduce techniques for decomposing compounds into fragments, algorithms for combining fragments (synthesis of molecules), and the steps by which we construct a molecule in the form of a molecular hypergraph.

5.1 Significance of the Problem

There is an urgent need for new antibiotics. Although the multidrug-resistance in pathogens is growing fast, the number of new drugs being developed to treat bacterial infections has reached its lowest point since the beginning of the antibiotic era [15, 79]. The resistance is particularly problematic in Gram-positive organisms S. aureus, E. faecalis and S. pneumoniae as well as a number of Gram-negative organisms including K. pneumonia, A. baumannii, and P. aeruginosa [72]. Hence, there is a dire need to develop new platforms and approaches to discover antibacterial agents against novel molecular targets. Not only new drugs are not being created, but also the existing process of creating drugs is slow and inefficient. Therein lies our innovation that makes this process more efficient.

Since fatty acids are only used for membrane biogenesis in bacteria, the enzymes of the fatty acid biosynthetic pathway have been identified as attractive targets for the development of novel antibacterial agents [46, 18, 88]. Bacterial biotin carboxylase (BC) is one portion of acetyl-CoA carboxylase (ACCase), a multifunctional enzyme complex that catalyzes the committed and regulated step in fatty acid biosynthesis. This metabolic pathway in bacteria is critical for several important biological processes including the synthesis and maintenance of cellular membranes. Scientists at Pfizer discovered several antibiotics against BC that belong to three different classes: pyridopyrimidines [65], amino-oxazoles [66] and the benzimidazole carboxamides[65]. Notwithstanding a great success of this structure-based design, all BC inhibitors developed to date show antibacterial activity only against Gram-negative organisms, while exhibiting either limited or no activity against

Gram-positive species. Therefore, novel broad-spectrum antibiotics against this promising molecular target remain to be discovered.

Due to extremely high costs of high-throughput screening, many drug discovery projects commonly employ inexpensive computations to support experimental efforts. In particular, *virtual screening*, a technique that shows great promise for lead discovery, has become an integral part of modern drug design pipelines. Here, the idea is to considerably reduce the number of candidate compounds that need to be tested experimentally against a protein target of interest. Due to advances in computer technology resulting in constantly increasing computational power, virtual libraries comprising many thousands of compounds can be rapidly evaluated in silico prior to experimental screens and at a fraction of the cost. Virtual screening approaches, historically divided into ligand- and structure-based algorithms prioritize drug candidates by estimating the probability of binding to the target receptor [57]. Among many methods developed to date, docking-based techniques are valuable tools for lead identification [28]. These algorithms rank compounds by predicting the binding mode for a query molecule in the binding pocket of the target protein, followed by the prediction of binding affinity from molecular interactions. There are many examples of a successful application of virtual screening tools to develop compounds with desired bioactivities [20, 81].

Computer-aided drug discovery traditionally utilizes large compound libraries for virtual screening. For example, the ZINC database is one of the most comprehensive repositories of commercially available compounds for virtual screening [48]. It currently features over 35 million compounds in ready-to-dock formats. These large generic collections of low molecular weight organic compounds provide a sufficient diversity to perform virtual screening against any molecular target, however, the vast majority of compounds will have a very low probability to exhibit the desired bioactivity for a specific target protein. Furthermore, considering the imperfections of compound ranking by virtual screening algorithms [55], even a large top-ranked subset of compound library may contain few active molecules.

Thus, the chances to identify novel, high-quality leads from large compound repositories are low. For instance, an internal analysis of the Abbott compound collection suggested that less than 4% of the compounds in their repository have the potential to yield novel kinase hinge-binders [6]. In order to address these issues, there have been significant efforts to augment existing collections with large numbers of compounds that have a higher potential for binding to specific targets of interest. Consequently, the trend in library design has shifted to include target class focusing in addition to diversity and drug-likeness criteria [59].

A target-focused library is a screening collection of compounds specifically tailored to modulate the function of a particular target or a protein family [9, 70]. There are a variety of approaches that have been developed for the design of target-specific *focused libraries* against, e.g. protein kinases, ion channels, G-protein coupled receptors (GPCRs), nuclear receptors, and protein-protein interfaces. Interestingly, these libraries not only reduce waste by eliminating compounds that are unlikely to bind to the target proteins, but often lead to an increase in the potency or specificity of binders, as demonstrated for c-Src kinase [60]. Several approaches employ molecular docking to determine target-specific thresholds that can be used as filters in virtual screening. This strategy was experimentally validated on the kinase-targeted library of 1,440 compounds and 41 kinases from five different families, demonstrating a 6.7-fold higher overall hit enrichment compared to a generic compound collection [42]. Furthermore, a structure-based modeling was used to create a small focused library against C. pneumoniae, a common pathogen recently linked to atherosclerosis and risk of myocardial infarction [11]. The experimentally determined hit rate for the targeted library was 24.2%, which is considerably higher than what would be expected for a generic library. Similar to structure-based approaches, ligand-based techniques can also be used in the focused library design, as shown for the GPCRs family [59]. Compared to large, diverse screening libraries, using relatively small, targeted collections significantly improves the odds of finding potential drug candidates, thus further reduces the costs of drug discovery.

Target-focused libraries are either designed or assembled based on some understanding of a specific protein target or a protein family. These collections are often compiled from larger, more diverse libraries using either molecular docking (structure-based approach) or ligand fingerprint similarity (ligand-based approach). The former employs structural, sequence and mutagenesis data, whereas the latter is based on the bio-molecular properties derived from known ligands, offering a useful way of "scaffold hopping" from one ligand class to another [73]. Target-focused libraries are often constructed around a single scaffold with one or more positions used to attach various chemical moieties or side chains. Although this approach can result in millions of different compounds [17], the chemical space remains largely unexplored, therefore truly novel compounds will not be discovered. On the other hand, combinatorial chemistry methods can produce a vast collection of divers compounds, so vast that only a tiny fraction of it could be explored, even using supercomputers. One can hardly imagine screening the chemical universe containing from 10^{12} to 10^{180} drug-like compounds [41]. Therefore, techniques for the design of libraries that populate the chemical subspace covering regions that are relevant to biology [30] are desperately needed. These methods hold a promise to contribute to the advancement of our knowledge of biological processes leading to new strategies to treat diseases.

Focused library design by molecular synthesis is essentially a combinatorial problem that can be addressed using graph theory. These techniques have been already extensively used in Computer Science and Artificial Intelligence (AI) for the synthesis of plans [40], problems and solutions in geometry [7, 8], hardware from specifications [78], and protocols [4, 71]. Graph-based approaches also have a wide range of applications in drug discovery including the analysis of chemical structures to better understand the common features present in drug molecules [12], the design of novel bioactive compounds with desired pharmacological profiles [36], structure-based modeling of protein flexibility upon ligand binding [19], the investigation of systems-level drug-target interaction networks [67], and drug repositioning [43].

Molecular bonding can be represented as a hypergraph whose nodes are molecular fragments and hyperedges represent molecular combinations that follow the laws of chemical bonding. The traversal algorithm filters the chemical space using additional assumptions about molecular properties and heuristics to restrict the search to relevant molecules. A path in the hypergraph from a set of source nodes to a target node represents a sequence of reactions that can lead to the formation of a complex molecule from constituent fragments. Hypergraph-based algorithms guarantee that all possible compounds will be generated and evaluated. We consider these details in the remainder of this chapter.

5.2 Molecular Fragments

Many focused collections of compounds for drug development have been compiled by industry from the results of high-throughput screens collected over years of experiments. However, these libraries often cover only a very limited repertoire of drug targets that are of interest to a particular company and are not available to the general scientific community. Existing computational methods for the construction of focused libraries are not only limited to the derivatives of already discovered scaffolds, but also designed for specific proteins, thus may not be generalized to a broad range of molecular targets. In contrast, our approach offers a unique capability to deliver high-quality focused libraries for a broad range of target proteins. Specifically, we include in this discussion our approach to compound decomposition into *molecular fragments*. Decomposition offers an easy way to create new chemical entities. Organic compounds are composed of sets of connected rigid fragments, essentially different ring structures, and flexible linkers with a varying number of rotatable bonds. Such description allows for the decomposition of any molecule into its building blocks tracking the atomic connectivity, so that new, chemically feasible molecules can be easily generated from molecular fragments.

Extraction of Molecular Fragments. Before describing the algorithm for molecular decomposition into fragments, we first define a few terms.

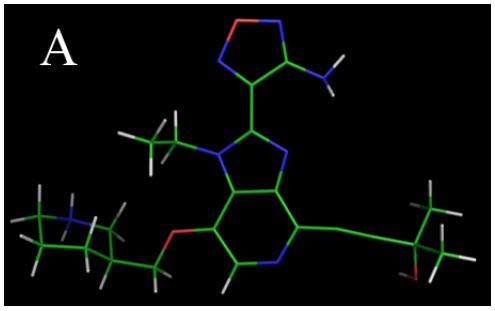


Figure 5.1: Fragment Extraction: A Bioactive Molecule

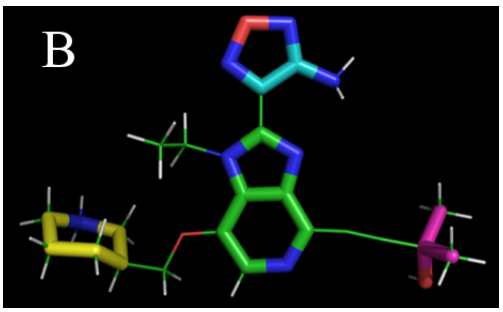


Figure 5.2: Fragment Extraction: Rigid Fragments (thick polygons) Identified

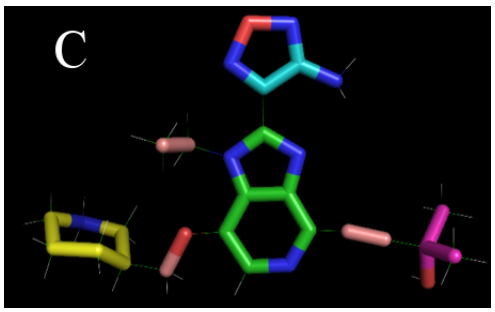


Figure 5.3: Fragment Extraction: Remaining Parts are Linkers (thick lines)

Input: A molecule M represented by atoms and chemical bonds.
Output: A unique set of constituent molecular fragments for M.

1. Identify all rotatable bonds in M.
2. Identify all rigid moieties of M.
3. Extract the remaining parts of M as flexible linkers.
4. Delete identical moieties.

Figure 5.4: Algorithm for Extracting Molecular Fragments from a Molecule

Definition 39. *A* rigid fragment *(rigid) is a set of at least four non-hydrogen atoms connected through non-rotatable bonds.*

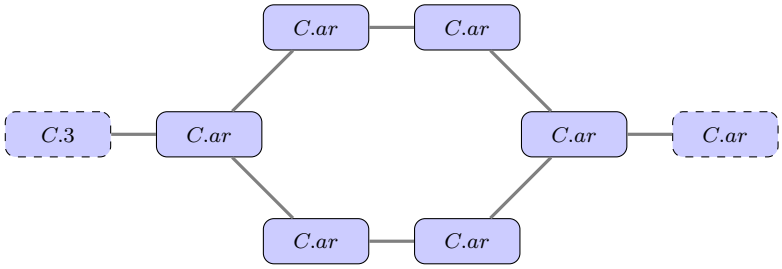


Figure 5.5: Sample Rigid Fragment

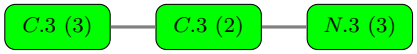


Figure 5.6: Sample Linking Fragment

Definition 40. *A* linking fragment *(linker) is a flexible fragment composed of a set of atoms connected through rotatable bonds.*

We now describe the algorithm stated in Figure 5.4 that decomposes chemical compounds into molecular fragments. In Step (1), We start with a molecule represented by atoms and chemical bonds as shown in Figure 5.1. Since rigids and linkers are defined based on rotatable bonds, the first step is identify such bonds in the given molecule. Since rigid fragments are closed sets of atoms, Step (2) extracts all of the rigids from the molecule as shown in Figure 5.2. Knowing what atoms have already been extracted as rigids, what remains in Step (3) are flexible linkers as shown Figure 5.3. It is clear that rigids provide structure to the molecule and linkers provide connectivity. Furthermore, in order to properly bond fragments using graph-based algorithms, we track the connectivity between individual fragments so that chemically feasible compounds can be synthesized. Last, in Step (4), the redundancy is removed from molecular fragments extracted from multiple compounds by deleting identical moieties.

This approach to molecular fragment extraction is fast (linear in the number of atoms and bonds in a given molecule) and is capable of processing large datasets of chemical compounds.

A sample rigid and linking fragment are depicted in Figure 5.5 and Figure 5.6, respectively. For a rigid fragment, we specify all constituent atoms in bold outline. All possible single bonds in a rigid fragment are specified with atom types surrounded with a dashed outline. In Figure 5.5, there are six carbon-aromatic atoms (C.ar), two of which can have a single bond, one to a carbon-3 (C.3) and one to a carbon-aromatic. For each atom in a linking fragment, the atom type of each atom is specified as well as the number of possible bonds in parentheses. The linking fragment in Figure 5.6 contains three atoms, the left-most atom has atom type carbon-3 (C.3) and can connect with up to three carbon-3 atom types.

5.3 Synthesis

We formalize molecular bonding over a given set of rigid and linking fragments restricted by the laws of chemistry. Molecular synthesis is a multi-phase process. First, we use a fixed-point approach to generate the complete set of molecules. Next, we identify a particular molecule of interest based on user input. Last, based on the input fragments and target molecule, we construct a molecular hypergraph. This molecular hypergraph can then be traversed to extract reachability and hyperpaths accordingly. In total, the synthesis of a molecule is not just the resultant molecule, it is the exact sequence of steps by which the molecule was generated.

5.3.1 Algorithms

A fragment-based approach to synthesis can theoretically result in an infinite molecular search space. By stating an upper bound based on molecule size, the synthesis process may still result in 10^8 molecules or more. It is therefore highly desirable to develop an efficient fixed-point algorithm for molecular synthesis that is complete; that is, all possible molecules that can be synthesized under chemical and physical constraints are guaranteed to be generated.

Algorithm 5.1 Complete, Level-Based Molecular Synthesis

```
1:  procedure SYNTHESIZE(L, R, max)        ▷ A set of linkers L, set of Rigids R, max
2:      Set⟨Molecule⟩M[max] ← {∅}          ▷ An array of unique molecules for each level
3:      F ← M[1] ← L ∪ R
4:      for ℓ = 2 to max do
5:          for all f ∈ F do
6:              for each m ∈ M[ℓ − 1] do
7:                  M[ℓ].AddAll(f.Compose(m))
8:              end for
9:          end for
10:     end for
11:     return ⋃_{ℓ=2}^{max} M[ℓ]
12: end procedure
```

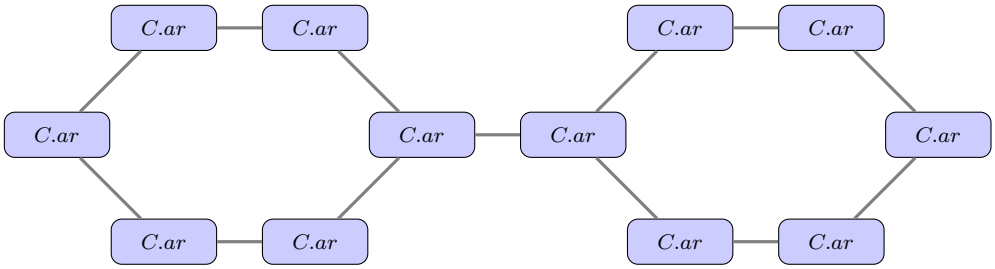


Figure 5.7: 2-Molecule Resulting From COMPOSE(R, R) with Rigid R from Figure 5.5

For expository purposes, we will refer to a k-molecule as a molecule that is composed of k molecular fragments. The algorithm in Algorithm 5.1 uses a level-based approach to molecular synthesis where all molecules in a level are composed of the same number of fragments.

In Algorithm 5.1, Line 3 initializes the synthesis process by storing the 1-molecules (i.e. fragments) into the array M (at index 1). In Line 4 to Line 10, we exhaustively synthesize each new level of molecules from the 2-molecules to max-molecules where max is the upper bound parameter set by the user. For simplicity, we store all of the k-molecules at index k in M. The low-level synthesis process is performed by the COMPOSE(m_1, m_2) function which takes two molecules m_1 and m_2 then combines them together in all possible orientations. Figure 5.7 depicts the result of calling COMPOSE(R, R) for the rigid fragment R from Figure 5.5. In this case, there is only one possible way to compose R with itself. For

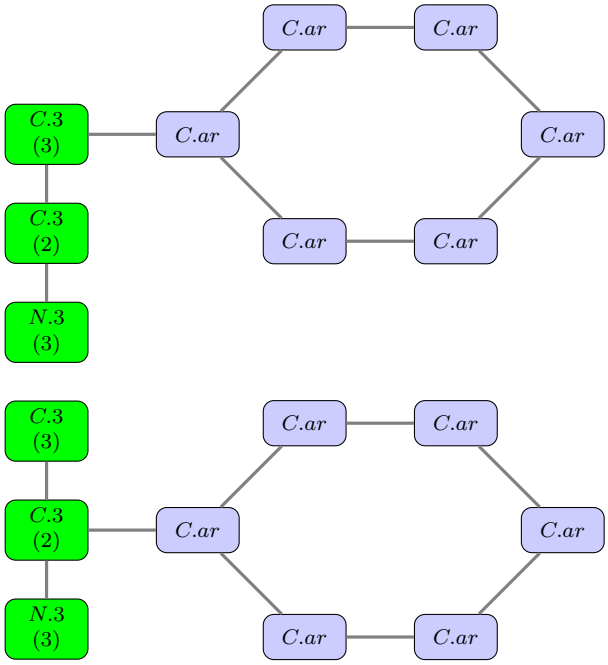


Figure 5.8: 2-Molecules From COMPOSE(R, L) with Rigid R from Figure 5.5 and Linker L from Figure 5.6

rigid fragment R from Figure 5.5 and linking fragment L from Figure 5.6, COMPOSE(R, L) results in the molecules shown in Figure 5.8.

Hypergraph construction is an exhaustive process in which two molecules (nodes) will be combined with a single bond to create a molecule. That is, if two source molecules may bond together two atoms to construct a target complex molecule, the bond is created and a new molecule is formed. This process describes the notion of a two-to-one source-to-target edge in the molecular hypergraph and the systematic construction of all possible complex molecules that may be formed from the input set of rigids and linkers.

`Compose` returns a set of molecules that meet the stated constraints, including Lipinski compliance [84] and added to the appropriate set of k-molecules. Last, on Line 11, we combine the sets of all synthesized molecules into a single collection that is to be returned.

The level-based approach described in Algorithm 5.1 is malleable depending on computational constraints. For example, Algorithm 5.1 implies that a level k must complete before level $k + 1$ starts. However, an astute observer will recognize that Algorithm 5.1

Algorithm 5.2 Bounded, Level-Based Molecular Synthesis

```
1:        ▷ Input: sets of linkers and rigids, upper bound of MAX-molecules to synthesize.
2:        ▷ Output: Collection of synthesized molecules.
3:  procedure SYNTHESIZE(ℒ, ℛ, MAX)
4:      Set⟨Molecule⟩M[max] ← {∅}                          ▷ Molecule accumulator
5:      Set⟨Molecule⟩W[max] ← {∅}                                  ▷ Worklist
6:      W[1] ← ℱ ← ℒ ∪ ℛ                          ▷ Initialize 1-molecules as fragments
7:      while ¬W[1].empty() do                  ▷ Empty W[1] ⇒ all levels ≥ 2 complete
8:          SYNTHESIZEHELPER(1, W, ℱ, MAX, M)
9:      end while
                    max
10:     return ⋃      M[ℓ]
                   ℓ=2
11: end procedure
12:
13:       ▷ SynthesizeHelper: inductively construct under bound constraints: ℓ → ℓ + 1
14: procedure SYNTHESIZEHELPER(ℓ, W, ℱ, MAX, M)
15:     if ℓ > MAX then                    ▷ Adhere to upper bound on level ℓ-molecules.
16:         return
17:     end if
18:     while ¬W[ℓ].empty() do
19:                              ▷ Check level ℓ + 1 capacities; process level ℓ molecules, if any
20:         while ¬W[ℓ + 1].atCapacity() and ¬W[ℓ].empty() do
21:             m ← W[ℓ].pop()                          ▷ Acquire molecule to process
22:             M[ℓ].Add(m)
23:             for all f ∈ ℱ do              ▷ Compose all ℱ with m: level ℓ → ℓ + 1
24:                 W[ℓ + 1].AddAll(f.Compose(m))
25:             end for
26:         end while
27:         SYNTHESIZEHELPER(ℓ + 1, W, ℱ, MAX, M)                ▷ Process level ℓ + 1.
28:     end while
29: end procedure
```

can easily be modified for a multi-threaded approach in which level k is a producer for level $k + 1$, the consumer. Thus, if each level maintains a thread acting as producer and consumer, synthesis can be expedited.

Similarly, we may introduce a bounded alternative of Algorithm 5.1. In Algorithm 5.2, we maintain an array of worklists (Line 5), one for each level that has an explicit capacity. If we reach the capacity of a worklist at level ℓ, we forgo processing the remaining items at level ℓ and inductively complete processing of all molecules at level $\ell + 1$ (Line 20). Otherwise, in Line 23 to Line 25 we compose a molecule from level ℓ with all of the fragments into

level $\ell + 1$ as before. We note that the approach in Algorithm 5.2 is appropriate for either serial or parallel syntheses depending on availability of computational power.

5.3.2 Molecular Filtration with Bloom Filters

Synthesis of molecules is limited by physical restrictions on molecules, but moreso time and space. An efficient synthesis must overcome time and space considerations, generate molecules within the physical restrictions, but do so without redundancy.

Using either Algorithm 5.1 or Algorithm 5.2 results in significant redundancy in synthesized molecules. The typical synthesis scenario from a basis of fragments will generate hundreds of millions of molecules which makes storing these molecules in memory infeasible; eliminating molecular redundancy requires a memoryless technique; synthesis requires a series of Bloom filters [14].

A Bloom filter is a probabilistic data structure that is efficient in terms of time and space. Although Bloom filters are well-studied, we describe their use in our synthesis domain. The main purpose of a Bloom filter is to determine whether an element is in a given set. Let $\mathcal{M}$ be a set of molecules and M a molecule. A Bloom filter is guaranteed to answer the query $M \in \mathcal{M}$ if molecule M is in set $\mathcal{M}$. Since a Bloom filer is a probabilistic data structure, it is subject to false positives: a query returns $M \in \mathcal{M}$ when $M \notin \mathcal{M}$. Fortunately, the rate of false-positives can be controlled.

While we omit some details of a Bloom filter, we consider the rate of false-positives. A Bloom filter is based on b the number of bits in the filter array, the number of distinct hash functions h, and the number of elements n we expect to insert into the filter. Assuming all hash functions hash elements uniformly to all b bits in the target array, the rate of false-positives for an element M not in a set $\mathcal{M}$ is given by $P\left(M \notin \mathcal{M}\right) = \left(1 - e^{-\frac{n \cdot h}{b}}\right)^h$. It can be shown that to minimize the rate of false-positives, the required number of hash functions h is given by $h = \frac{b}{n} \cdot \ln 2$. If p is the desired false-positive rate, it can also be shown that the required number of bits $b = -\frac{n \ln p}{(\ln 2)^2}$ [14].

Consider a molecular Bloom filter F in which we tolerate a 1% false-positive rate for 10^8 molecules. In this case, we require $b = 9.585 \cdot 10^8 \approx 120$ megabytes with $h = 7$ hash functions. This means each addition of a molecule to F and each query on F is subject to the worst case $O(h) = 7$ hashings.

Molecular synthesis requires a string representation of molecules. In particular, a molecule M is represented using the SMILES specification [3] as a molecular fingerprint as input to each Bloom filter. We can modify the Compose function in Algorithm 5.2 by including several Bloom filters: a single, overall filter F as well as a filter F_ℓ for each level. When an ℓ-molecule M is synthesized, we first check whether it has been previously synthesized by querying F_ℓ. If the molecule has not been synthesized ($M \notin F_\ell$), we add M to F_ℓ and query F. If $M \notin F$, we add M to F and proceed as in Algorithm 5.2 by adding M to the level-ℓ queue to be processed into level-$(\ell + 1)$ molecules. Clearly, the global F requires the most memory, but ensures that molecules containing different number of fragments with the same SMILES representation are filtered as redundant.

5.4 Molecular Hypergraph

In this section we formalize the molecular synthesis space as a hypergraph, specifically, a *molecular hypergraph*. We begin with some definitions related to molecules and end with our definition of a *molecular hypergraph*.

5.4.1 Definitions

For a molecule M, $R(M)$ is the constituent set of unique rigid fragments and $L(M)$ the constituent set of unique linking fragments. For a set of molecules $\mathcal{M}$, $R(\mathcal{M}) = \bigcup_{M \in \mathcal{M}} R(M)$ the unique set of rigid fragments and $L(\mathcal{M}) = \bigcup_{M \in \mathcal{M}} L(M)$ the constituent set of unique linking fragments.

Molecular synthesis depends on a set of fragments we refer to as the *molecular basis* as defined in Definition 41.

Definition 41 (Molecular Basis and Cardinality)**.** *For a set of molecules $\mathcal{M}$, we refer to the simple molecular moieties composed of rigid fragments $\mathcal{R} = R(\mathcal{M})$ and linking fragments $\mathcal{L} = L(\mathcal{M})$ as the* molecular basis for molecular set *M, $\mathcal{B}_{\mathcal{M}}$, such that $\mathcal{B}_{\mathcal{M}} = \mathcal{R} \cup \mathcal{L}$. We also refer to the* cardinality *of a basis, $|\mathcal{B}_{\mathcal{M}}|$, as the number of unique fragments in $\mathcal{B}_{\mathcal{M}}$.*

We note that the cardinality of a basis set of molecules is a simple sum of the number of unique rigids and linkers: $|\mathcal{B}| = |R(M)| + |L(M)|$. We also note that for a rigid fragment R, $\mathcal{B}_R = \{R\}$, the singleton set containing only itself and thus $|\mathcal{B}_R| = 1$, similarly for linking fragments.

Our goal in molecular synthesis is to combine fragments into various combinations yielding molecules; we explicitly define the notion of complex molecule.

Definition 42 (Complex Molecule)**.** *A complex molecule is a molecule composed of two or more fragments; the fragments may or may not be unique.*

Clearly, any k-molecule where $k \geq 2$ is a complex molecule. We note that in some special cases it is possible for a complex molecule, C, be composed of copies of a single fragment; hence, $|\mathcal{B}_C| = 1$.

We also define notation for the number of fragments of a k-molecule.

Definition 43 (Cardinality of a Molecule)**.** *A k-molecule is composed of k, possibly non-unique fragments. We say that for a molecule M, $|M| = k$.*

5.4.2 The Molecular Hypergraph

We use a hypergraph-based approach to synthesis of complex molecules. Each node in the hypergraph represents a molecule, either rigid, linker, or complex molecule. When two molecules (source nodes) can be combined with a single, target bond to create a molecule, the hypergraph will contain a corresponding hyperedge as defined in Definition 44.

Definition 44 (Attraction and Repulsion Hyperedges)**.** *We refer to a hyperedge in which two source molecules are combined into a complex, target molecule as an* attraction hy-

peredge. *Similarly, a* repulsion hyperedge *has two source nodes in which one molecule is subtracted from the other complex molecule resulting in a simpler molecule.*

Given three molecules S_1, S_2, and T that constitute an attraction hyperedge $\{S_1, S_2\} \xrightarrow{A} T$, we conversely have two repulsion hyperedges $\{S_1, T\} \xrightarrow{R} S_2$ and $\{S_2, T\} \xrightarrow{R} S_1$. In these two cases, we annotate each hyperedge with the type of hyperedge: repulsion (R) or attraction (A).

Having defined the correspondence in the molecular synthesis space with nodes and hyperedges, we may now define the molecular hypergraph.

Definition 45 (Molecular Hypergraph)**.** *A* molecular hypergraph $H(\mathcal{M}, \mathcal{E}_{\mathcal{A}})$ *is a synthesis hypergraph in which* $\mathcal{M}$ *is the set of molecules (nodes) and* $\mathcal{E}$ *is the set of hyperedges over a set of bond-based annotations* $\mathcal{A}$*. We say that* $H(\mathcal{M}, \mathcal{E}_{\mathcal{A}})$ *is the* molecular hypergraph corresponding to the molecular synthesis of basis $\mathcal{B}_{\mathcal{M}}$ *and note that* $\mathcal{B}_{\mathcal{M}} \subseteq \mathcal{M}$*. Each hyperedge* $E \in \mathcal{E}_{\mathcal{A}}$ *is of the form* (S, t, A) *where* $S \subseteq \mathcal{M}$ *is a set of molecules,* $t \in \mathcal{M}$*, and* $A \in \mathcal{A}$*.*

The set of bond-based annotations $\mathcal{A}$ is a parameterized set of expressions defined by the user. For example, in the case where a user wishes to omit all repulsion edges, the associated set of annotations will lead to a subset of all hyperedges that meet the defined characteristics.

5.5 On-Demand Molecular Hypergraph Construction and Traversal

Without an upper bound placed of the number of fragments in a molecule, Algorithm 5.1 results in a complete synthesis of the entire molecular synthesis space; however, computational and memory limitations prohibit construction of a corresponding molecular hypergraph. Given a molecule M_t and the basis of M_t, $\mathcal{B}_{M_t}$, we construct the corresponding *template*-based molecular hypergraph $H(\mathcal{M}, \mathcal{E}_{\mathcal{A}})$. In order to construct a template-based molecular hypergraph, we use a fixed-point version of Algorithm 5.1 as defined in Algorithm 5.3.

Algorithm 5.3 Template-Based Hypergraph Construction

```
1:                                          ▷ Linkers 𝓛, Rigids 𝓡, Template Molecule M_t
2: procedure TemplateConstruct(𝓛, 𝓡, M_t)
3:     Hypergraph G
4:     Queue⟨Molecule⟩ W ← 𝓛 ∪ 𝓡
5:     while ¬W.empty() do
6:         c ← W.dequeue()
7:         G.addNode(c)
8:         for all m ∈ G do
9:             for all t ∈ Compose(m, c) do
10:                if |M_t| > |t| then
11:                    W.Add(t)
12:                    G.AddNode(t))
13:                    G.AddHyperedge((m, c), t)
14:                end if
15:            end for
16:        end for
17:    end while
18:    return G
19: end procedure
```

The result of Algorithm 5.3 is a hypergraph $H(\mathcal{M}, \mathcal{E}_{\mathcal{A}})$ with the following characteristics:

- All hyperedges are attraction edges.
- The nodes corresponding to fragments in $\mathcal{B}_{M_t}$ have no incoming hyperedges (they are leaves in H^T).
- The node corresponding to M_t has no outgoing hyperedges (root in H^T).
- M_t is the largest molecule in H. That is, for each $M \in \mathcal{M}$, $|M_t| > |M|$.

Algorithm 5.3 takes a constructivist perspective in which fragments are combined into larger and larger molecules. A converse, equivalent destructivist version of the algorithm might begin with molecule M_t by splitting it into smaller molecules and eventually fragments. This observation is due to the fact that a molecular hypergraph $H(\mathcal{M}, \mathcal{E}_{\mathcal{A}})$ with only attraction edges results in H^T being a directed acyclic graph (DAG).

Molecular synthesis is a process by which we construct all possible molecules from fragments, but more importantly, a molecular hypergraph provides the sequence of bonds

necessary to create a molecule. Let $H(\mathcal{M}, \mathcal{E}_{\mathcal{A}})$ be a molecular hypergraph using Algorithm 5.3 with input M_t and basis $\mathcal{B}_{M_t}$. Recall, each hyperedge in a molecular hypergraph is annotated according to user parameters. For H, it is possible to select a pebbled molecular hypergraph $H_P(\mathcal{M}, \mathcal{E}_{\mathcal{A}})$ according to Algorithm 2.1. Each hyperpath from $\mathcal{B}_{M_t}$ to M_t in H_P corresponds to a sequence of molecular bonds that may be replicated in a physical laboratory environment.

5.6 Experimental Results

The main experimental objective was to validate the molecular synthesis technique. We used two protocols for validation: self-reconstruction and cross-validation with leave-one-out testing to determine if *Synth* can generate novel molecules.

5.6.1 Self-Reconstruction

Validation is performed by (1) deconstructing fragments from active molecules and (2) running *Synth* to validate that the original, parent molecule is reconstructed from its own fragments. Failing at this step means that *Synth* was incapable of forming reasonable compounds.

We ran these steps on our set of active molecules from the database of useful decoys (DUD-E) [33], more than 20,000 chemical compounds in total. In these experiments we use the Tanimoto (similarity) Coefficient (TC) [85] as a heuristic to compare molecules; $0 \leq TC \leq 1$ where $TC = 0$ means no similarity and $TC = 1$ means absolute similarity.

In Figure 5.9, two different sets were used to evaluate the ability of *Synth* to synthesize a single molecule from its components. Specifically, the min info refers to a library of fragments where the linkers have bonding rules similar to rigids in which a connection may occur only from points in which there was an original connection. We compare this to the max info library where linkers are able to connect at any point that a hydrogen could attach. The min info library reduces the chance (lowers the probability) of synthesizing ill-structured molecules with short linkers attaching to a large rigids at every atom; for an

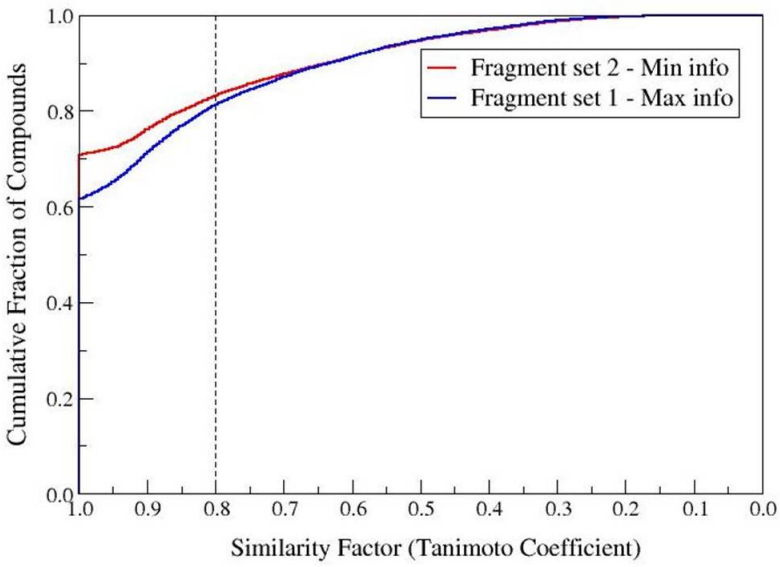


Figure 5.9: Reconstruction with Min and Max Info Libraries

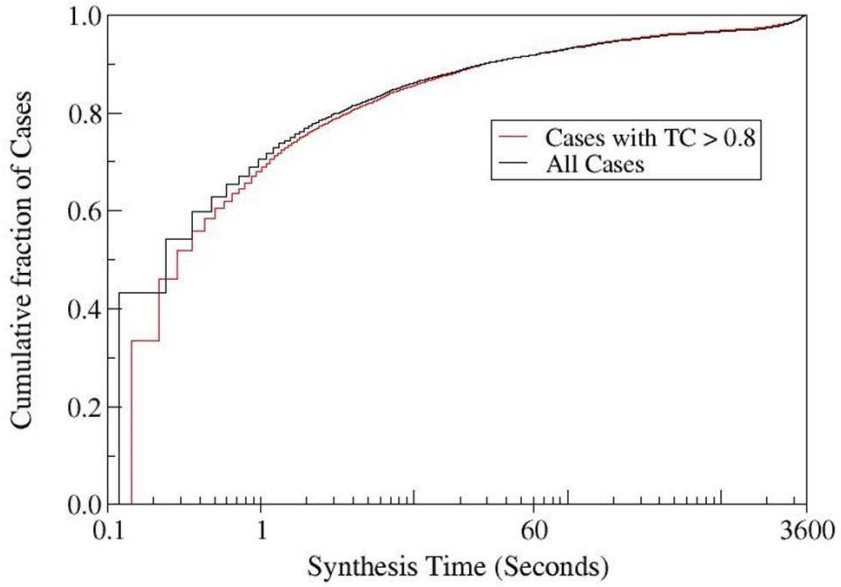


Figure 5.10: Cumulative Frequency of Time to Reconstruct Molecules

example of this phenomenon see Figure 5.15. Overall, 70% of the original molecules were recaptured with TC of 1.0, and more than 80% were synthesized to a very high degree of similarity ($TC > 0.8$).

Figure 5.10 demonstrates that the majority (90%) of the compounds were rebuilt with TC greater than 0.9 in less than a minute while half of the molecules took only a fraction

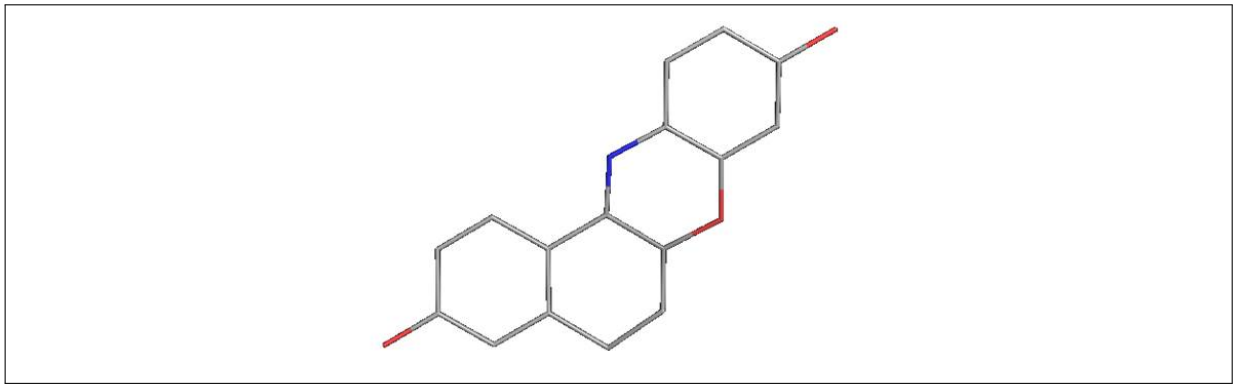


Figure 5.11: A Molecule Composed of a Single Rigid Fragment that Fails Reconstruction

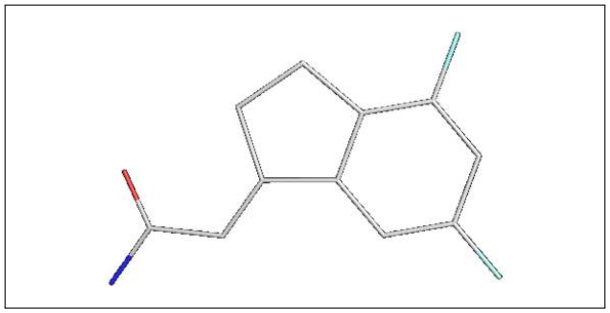


Figure 5.12: Original Molecule

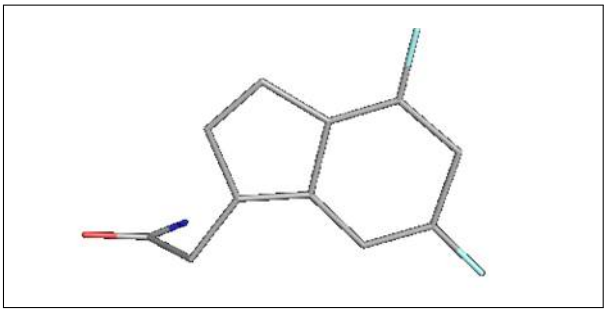


Figure 5.13: False-Negative Corresponding to Figure 5.12

of a second to be reconstructed. In the end, this means that *Synth* is accurate, efficient, and can explore deep in the chemical compound search space in a short amount of time.

Failed Reconstruction of Molecules. There are three reasons *Synth* did not regenerate every parent molecule from its fragments; we describe each in turn.

The first failed case is attributed to a molecule that is composed of one solid, rigid fragment. For example, the molecule in Figure 5.11 is composed of a single, rigid fragment composed of four hexagons. If we reconstruct the molecule in Figure 5.11, no connection points exist for further bonding. Hence, no further fragments can bond with this fragment and the result of synthesis would be one molecule, the molecule itself composed of a single rigid.

The second failure in reconstruction is not attributed to the fragmentation nor synthesis process, it is a result of using the open-source chemical toolbox, Open Babel [68]. Given

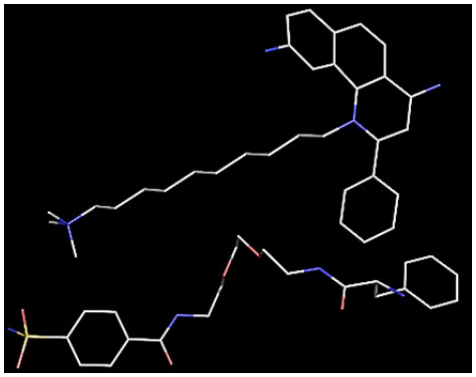


Figure 5.14: Original Molecule

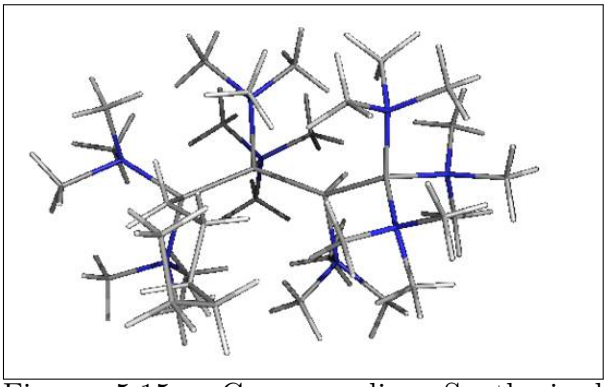


Figure 5.15: Corresponding Synthesized Molecule with Large Probability Space

the molecule in Figure 5.12, we synthesized the corresponding molecule in Figure 5.13 which differs only in the 3-dimensional coordinates of some atoms. This is a case of a false negative in the Open Babel similarity search. That is, Open Babel does not recognize the two molecules (Figure 5.12 and Figure 5.13) as perfectly similar ($TC = 1.0$), rather Open Babel calculates $TC = 0.8$ and that is not accurate upon inspection in a visual environment. *Synth* successfully synthesized the parent molecule in this case, but the similarity assessment protocol fails to recognize the equality.

The last synthesis scenario that fails arises when we compare the original molecule in Figure 5.14 to the synthesized version in Figure 5.15. This situation was described previously when discussing min / max info libraries. In this case, the molecule in Figure 5.15 contains many short linkers and is saturated with large groups or fragments at every single atom. Linkers do not function as thus in practice; hence, such molecules are not plausible due to steric hindrance (large groups prevent reactions we might observe in a related molecule with smaller groups).

5.6.2 Cross-Validation

We use cross-validation to determine if *Synth* can synthesize novel molecules, significantly different from their original parent molecules. Cross-validation is performed using a scenario. In this context a scenario involves clusters of active compounds that are parti-

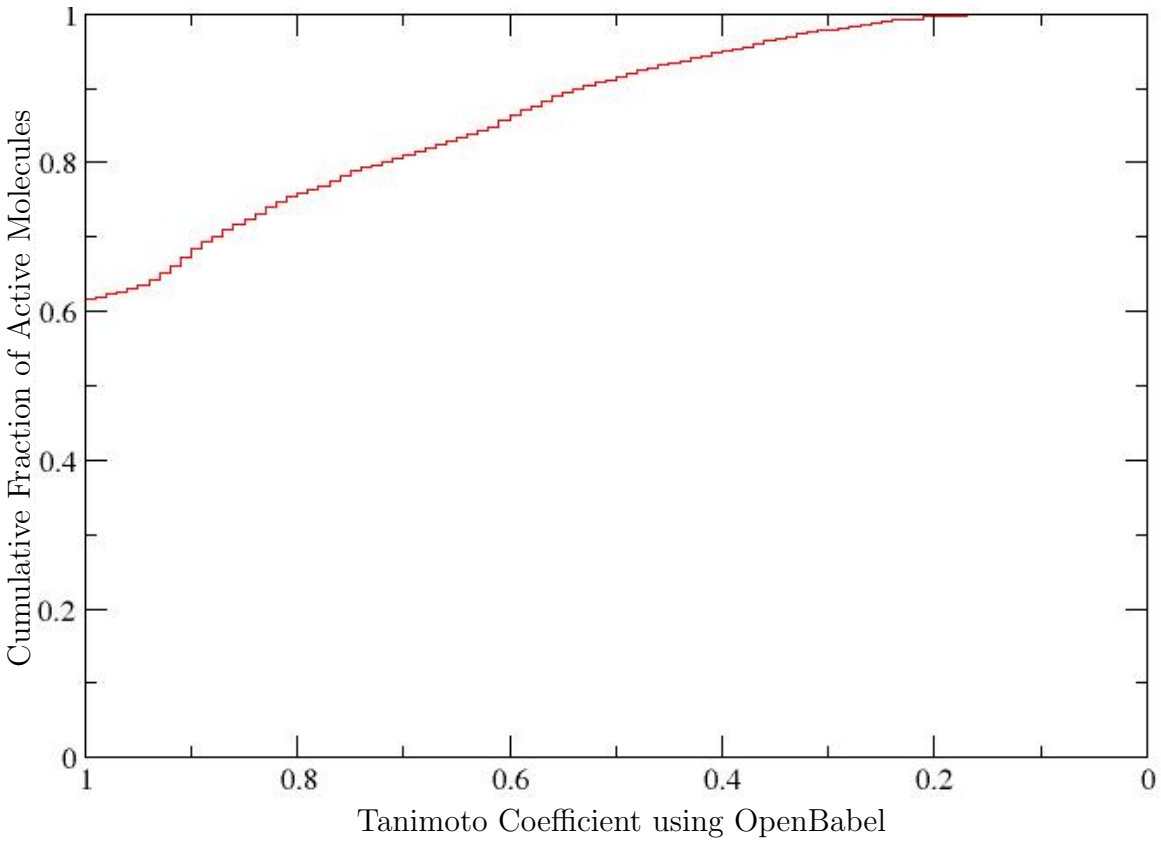


Figure 5.16: Self-Reconstruction of Scenario Target Compounds with $TC < 0.5$

tioned as follows: one cluster represents the *target* set of up to 3 active compounds while the remaining clusters of active compounds are defragmented and passed to *Synth*. The goal is verify whether *Synth* generated the active compounds in the target cluster.

Cross-validation was implemented using a fast search protocol via OpenBabel to collect only those molecules with $TC > 0.5$ compared to an active compound in the target cluster. As a second similarity measure, we take the set of 'similar' molecules acquired from OpenBabel and reconstruct the 3D coordinates of the atoms using *obgen* [80] and invoke *kcombu* [53, 54] to compare with the target active compounds.

20,000 scenarios were constructed and executed. In 8,000 of these scenarios, synthesis resulted in $TC \leq 0.5$ of target compounds with OpenBabel in the 'first' round. In these cases we need to verify that the target compounds were significantly different than the constituent fragments that would be used to construct it; that is, we verify our algorithms properly synthesize since the information to build these target compounds did not exist

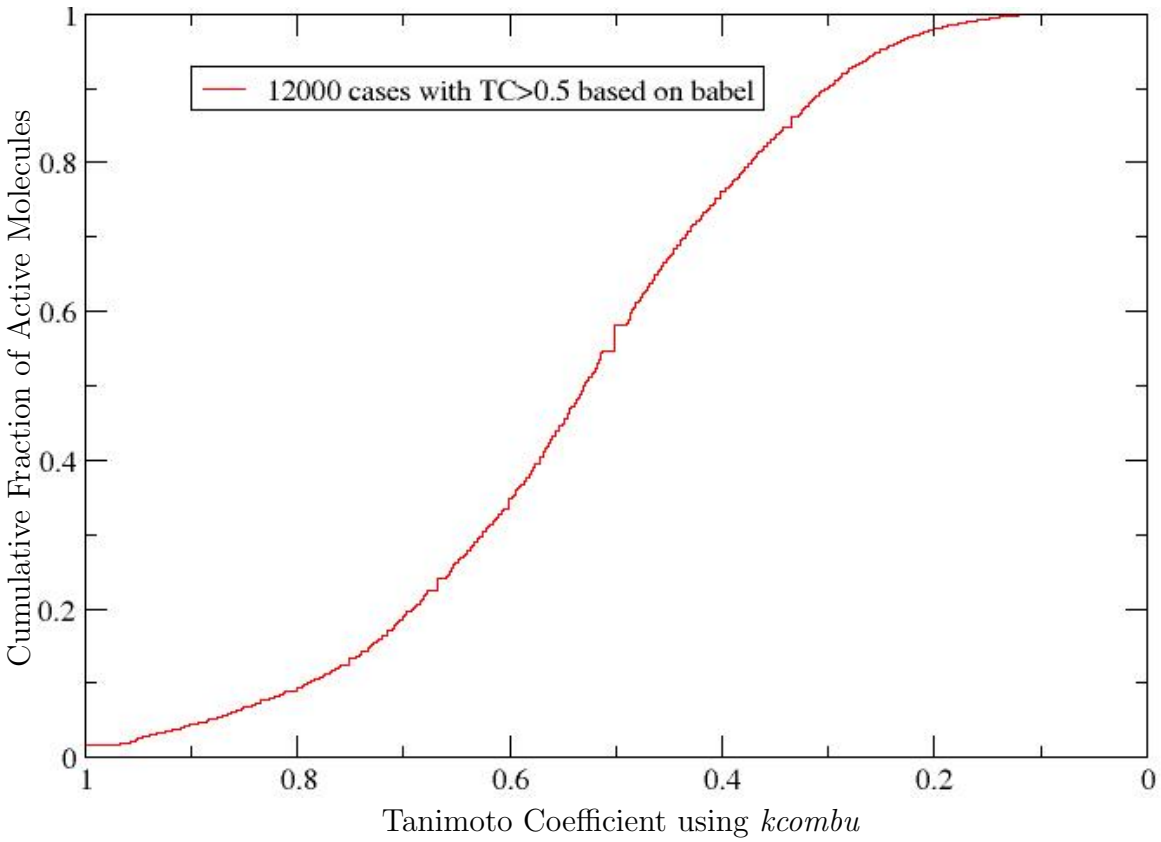


Figure 5.17: Self-Reconstruction of Scenario Target Compounds with $TC > 0.5$

in the other clusters used for synthesis. In Figure 5.16, we observe that the target 8,000 molecules were subjected to self-reconstruction (§5.6.1) using *Synth*: 80% of molecules are reconstructed with $TC > 0.7$ on par with our self-reconstruction analyses.

The remaining 12,000 scenarios with $TC > 0.5$ are represented in Figure 5.17 where all of the target molecules have somewhat similar molecules constructed by *Synth*: more than 50% of the target molecules have a corresponding synthesized molecule with $TC > 0.5$. This is significant since the constituent fragments of the target molecules were not available in the synthesis process, yet similar molecules were constructed.

Last, we compare similarity measures using *kcombu* and Openbabel in Figure 5.18. From Figure 5.18 we see that *kcombu* is more strict in predicting similarity. This is an important observation as we can say that our approach to select target molecules with $TC > 0.5$ using the OpenBabel 'fast' search followed by *kcombu* for refinement does not

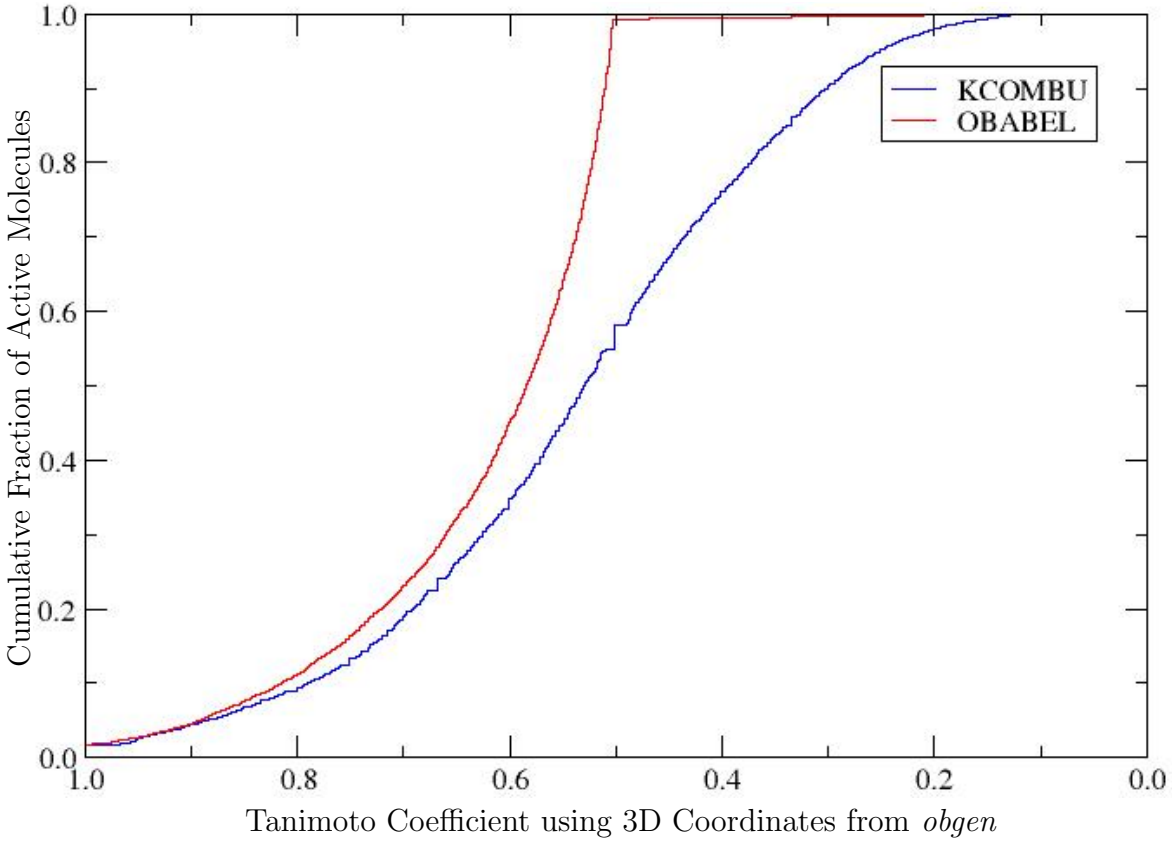


Figure 5.18: Comparing Similarity with Scenarios using *kcombu* and OpenBabel

lose any molecules with $TC < 0.5$. Specifically, using OpenBabel captures everything with $TC > 0.5$ (and more).

5.7 Related Techniques

There are two techniques that are often used in search for developing target proteins: (1) protein redesign [83] and from-scratch (de novo) protein design.

In protein redesign, some mutation of amino acids occurs while most residues in the sequence are maintained—the backbone. [31] is a protein redesign suite that improves flexibility of the protein backbone and models proteins and ligands as ensembles of low-energy structures as the K* algorithm [58]. From there, [31] performs globally optimal protein design search with respect to an input model. Comparatively, our synthesis technique constructs the entire protein search space in the form a library as a de novo protein design since fragment-based construction is not based on a previous sequence.

Synth is one example of de novo protein design. Some of the best-known software implementations for de novo construction using fragment-based assembly techniques include Fragfold [50], Simfold [38], and Rosetta [25]. Each differ in computational techniques in energy functions, fragment size, and heuristics. [45] uses Rosetta to investigate the de novo construction with a focus on fragment length and move size (an insertion-based approach). Specifically, [45] considers fragment length in the range of 6 to 18. The fragment size in [45] is extremely coarse compared to our fragmentation techniques. Secondly, our synthesis technique is focused on constructing a library by exploring the entire synthesis space whereas folding techniques such as [45] use insertion operations to investigate candidate proteins.

Chapter 6
Conclusions and Future Work

We conclude this dissertation with our contributions and suggest avenues of continued research in synthesis, including a generalization of hypergraphs in the synthesis problem space.

6.1 Generalizing the Hypergraph Approach

Our hypergraph-based approach can be adapted to work for any domain where the goal is to derive a new fact (or even compute some desired value) using a series of steps starting from some set of facts. Using the notion that each step should involve deriving a new fact using previously known facts is applicable to a variety of non-inductive proof domains (including geometry, algebra, and logic). We also feel that we might be able to generalize our approach to some domains in physics including mechanics and electrical circuits where the solution is again a sequence of steps in many cases. In contrast, our approach will not work for domains where the solution is not a sequence of steps such as construction problems (including algorithms, automata, geometry constructions, etc).

6.2 Conclusions and Future Work in Geometry Problem Synthesis

In Chapter 3 we described and evaluated a semi-automated technique for geometry proof problem synthesis implemented as *GeoTutor*. In Chapter 4 we built on *GeoTutor* and presented algorithms in a tool called *GeoShader* that efficiently solves a given shaded area problem, synthesizes such shaded area problems for existing figures as well as fresh figures. Our work in geometry problem synthesis is a cross-disciplinary approach that combines ideas from computational geometry, logical reasoning, and search heuristics. Together, *GeoTutor* and *GeoShader* provide a computationally viable foundation for an intelligent tutoring system for Euclidean Geometry. Generating problems for assignments or exams is a difficult and tedious process for an educator and the gift of time for a teacher

is the most valuable asset to educating all children. Time means more individual attention for each student so that teachers can do what they do best: teach students.

While not all features of a formal intelligent tutor have been explored in this dissertation, both *GeoTutor* and *GeoShader* are the foundation for an intelligent tutoring system with respect to problem synthesis. Future work will include a formal implementation and investigation into automated generation of interesting assignments as a component of personalized workflow, complete exam generation for teachers, an interactive hint system, and interactive solution verification system for students.

In the future, we plan to deploy our tools in high schools and conduct user studies to understand its effectiveness in an educational environment by measuring its effectiveness in improving student learning. Other future work will involve a user study that will examine the utility of the figure synthesis techniques as well as our definitions of interesting and complete for both proof problems and shaded area problems.

We also would like to explore natural language generation. Our problem generation tool generates problems at the level of logical predicates. It would therefore be useful to translate the logical predicates into an equivalent, but succinct, natural language description in the form of a word problem.

With our work in geometry problem synthesis, we assumed all figures were embedded in the Euclidean plane and *drawn to scale*. Future work may consider figure mutability. Although most geometry books offer problems with figures that are drawn to scale, in reality, biology, physics, and mathematics sometimes depict imperfect diagrams and thus require pre-processing to handle scale. Simply, what you see is not always what you get. Overcoming mutability of figures is a significant, but important task for synthesis of realistic problems in any domain.

Last, we may also consider problem and solution synthesis of other mensuration problems such as area and volume in Calculus by extending our approach to handle functions.

6.3 Conclusions and Future Work in Molecular Synthesis

In Chapter 5 we described algorithms to perform synthesis of molecules based on molecular fragments. The synthesis and hypergraph construction algorithms were developed in tool *Synth*. We validated *Synth* by creating the molecular fragments for a set of more than 20000 molecules and then used *Synth* to regenerate an equivalent, original molecule in less than one minute for 90% of molecules and one hour for 100% of molecules.

Currently, *Synth* prunes the infinite search space in a limited manner using Lipinski compliance, probability pruning, and an upper bound to molecular weight. These pruning techniques are based mainly on limiting depth in the search space (e.g. we limit the number of constituent fragments of molecules); however, we currently do not limit the width of the search space. Future work will introduce template-based techniques for pruning the width of the search space; i.e. providing the user with a more targeted synthesis using user-specified parameters specified by the user. For example, we may introduce distance metrics to construct neighborhoods around target molecules focusing the synthesis by using proximity techniques. Such heuristics require corresponding features be integrated into *Synth*; as complexity increases, a visual interface is required for broad appeal in the biology and chemistry communities.

References

[1] http://www.wolframalpha.com/problem-generator/$#$/, 2014.

[2] http://www.autotutor.org/, 2014.

[3] http://www.opensmiles.org/, 2015.

[4] F. Afrati, C. Papadimitriou, and G. Papageorgiou. The synthesis of communication protocols. *Algorithmica*, 3:451–472, 1988.

[5] U. Z. Ahmed, S. Gulwani, and A. Karkare. Automatically generating problems and solutions for natural deduction. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, 2013.

[6] I. Akritopoulou-Zanze and P. Hajduk. Kinase-targeted libraries: the design and synthesis of novel, potent, and selective kinase inhibitors. *Drug Discov Today*, 14(5-6):291–7, 2009.

[7] C. Alvin, S. Gulwani, R. Majumdar, and S. Mukhopadhyay. Synthesis of geometry proof problems. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 245–252, 2014.

[8] C. Alvin, S. Gulwani, R. Majumdar, and S. Mukhopadhyay. Synthesis of solutions for shaded area geometry problems. In *Submitted to IJCAI*, 2015.

[9] H. and et al. The design and application of target-focused compound libraries. *Comb Chem High Throughput Screen*, 14(6):521–31, 2011.

[10] E. Andersen, S. Gulwani, and Z. Popovic. A trace-based framework for analyzing and synthesizing educational progressions. In *2013 ACM SIGCHI Conference on Human Factors in Computing Systems, CHI '13, Paris, France, April 27 - May 2, 2013*, pages 773–782, 2013.

[11] A. andet al. Similarity based virtual screening: a tool for targeted library design. *J Med Chem*, 49(7):2353–6, 2006.

[12] G. Bemis and M. Murcko. The properties of known drugs. 1. molecular frameworks. *J Med Chem*, 39(15):2887–93, 1996.

[13] C. Berge. *Graphs and hypergraphs*, volume 45. North-Holland Mathematical Library; ELSEVIER SCIENCE PUBLISHERS B.V., 1989.

[14] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[15] Boucher, H.W. and et al. Bad bugs, no drugs: no eskape! an update from the infectious diseases society of america. *Clin Infect Dis,*, 48(1):1–12., 2009.

[16] Boyd, Cindy J. and et al. *Geometry (NJ Edition)*. Glencoe / McGraw-Hill, New York, NY, 2006.

[17] M. Brylinski and G. Waldrop. Computational redesign of bacterial biotin carboxylase inhibitors using structure-based virtual screening of combinatorial libraries. *Molecules*, 19(4):4021–45, 2014.

[18] J. Campbell and J. J.E. Cronan. Bacterial fatty acid biosynthesis: targets for antibacterial drug discovery. *Annu Rev Microbiol*, 55:305–32, 2001.

[19] H. Carlson. Protein flexibility and drug design: how to hit a moving target. *Curr Opin Chem Biol*, 6(4):447–52, 2002.

[20] C. Cavasotto and A. Orry. Ligand docking and structure-based virtual screening in drug discovery. *Curr Top Med Chem*, 7(10):1006–14, 2007.

[21] CBSE, India, 2012. http://cbse.nic.in/.

[22] C.-L. Chang and R. C.-T. Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, Inc., Orlando, FL, USA, 1st edition, 1997.

[23] T. Chew. *Singapore Math Challenge (Grade 5+).* Frank Schaffer Publications, 2008.

[24] S.-C. Chou, X. shan Gao, and J.-Z. Zhang. *Machine proofs in geometry—Automated production of readable proofs for geometry theorems.* World Scientific, 1994.

[25] R. Consortium. The rosetta software — rosettacommons, 2015.

[26] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (3. ed.).* MIT Press, 2009.

[27] H. Coxeter. *Introduction to Geometry.* John Wiley, 1969.

[28] D.A. Gschwend, A.C. Good, and I.D. Kuntz. Molecular docking towards drug discovery. *J Mol Recognit*, 9(2):175–86, 1996.

[29] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, Berlin Heidelberg, third edition, 2008.

[30] C. Dobson. Chemical space and biology. *Nature*, 432(7019):824–8, 2004.

[31] B. Donald. Donald lab at duke university, 2015.

[32] W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *J. Log. Program.*, 1(3):267–284, 1984.

[33] Dude.docking.org. Dud-e: A database of useful (docking) decoys enhanced, 2015.

[34] D. Eberly. The minimal cycle basis for a planar graph, 2015.

[35] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, 1987.

[36] A. P. Estrada, E. and R. Garcia-Domenech. Designing sedative/hypnotic compounds from a novel substructural graph-theoretical approach. *J Comput Aided Mol Des.*, 12(6):583–95, 1998.

[37] C. for Disease Control, 2015.

[38] Y. Fujitsuka, G. Chikenji, and S. Takada. Simfold energy function for de novo protein structure prediction: Consensus with rosetta. *Proteins: Structure, Function, and Bioinformatics*, 62(2):381–398, 2006.

[39] X.-S. Gao and Q. Lin. Mmp/geometer a software package for automated geometric reasoning. *Automated Deduction in Geometry*, 2004.

[40] M. Ghallab, D. S. Nau, and P. Traverso. *Automated planning - theory and practice*. Elsevier, 2004.

[41] A. Gorse. Diversity in medicinal chemistry space. *Curr Top Med Chem*, 6(1):3–18, 2006.

[42] Gozalbes, R. and et al. Development and experimental validation of a docking strategy for the generation of kinase-targeted libraries. *J Med Chem*, 51(11):3124–32, 2008.

[43] Gramatica, R. and et al. Graph theory enables drug repurposing–how a mathematical model can drive the discovery of hidden mechanisms of action. *PLoS One*, 9(1):e84912, 2014.

[44] S. Gulwani, V. A. Korthikanti, and A. Tiwari. Synthesizing geometry constructions. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011*, pages 50–61, 2011.

[45] J. Handl, J. Knowles, R. Vernon, D. Baker, and S. C. Lovell. The dual role of fragments

in fragment-assembly methods for de novo protein structure prediction. *Proteins: Structure, Function, and Bioinformatics*, 80(2):490–504, 2012.

[46] S. W. Heath, R.J. and C. Rock. Lipid biosynthesis as a target for antibacterial agents. *Prog Lipid Res,*, 40(6):467–97, 2001.

[47] Holt, Rinehart, and Winston. *Holt Geometry: Homework and Practice Workbook*. Holt, Rinehart, and Winston, Orlando, FL, 2007.

[48] J. Irwin and B. Shoichet. Zinc–a free database of commercially available compounds for virtual screening. *J Chem Inf Model*, 45(1):177–82, 2005.

[49] S. Itzhaky, S. Gulwani, N. Immerman, and M. Sagiv. Solving geometry problems using a combination of symbolic and numerical reasoning. In *Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013. Proceedings*, pages 457–472, 2013.

[50] D. T. Jones. Predicting novel protein folds by using fragfold. *Proteins: Structure, Function, and Bioinformatics*, 45(S5):127–132, 2001.

[51] R. Jurgensen, R. Brown, and J. Jurgensen. *Geometry*. Houghton Mifflin Company, Boston, MA, 1988.

[52] D. Kapur. Using gröbner bases to reason about geometry problems. *J. Symb. Comput.*, 2(4):399–408, 1986.

[53] T. Kawabata. Build-up algorithm for atomic correspondence between chemical structures. *Journal of Chemical Information and Modeling*, 51(8):1775–1787, 2011.

[54] T. Kawabata and H. Nakamura. 3d flexible alignment using 2d maximum common substructure: Dependence of prediction accuracy on target-reference chemical similarity. *Journal of Chemical Information and Modeling*, 54(7):1850–1863, 2014.

[55] Kitchen, D.B. and et al. Docking and scoring in virtual screening for drug discovery: methods and applications. *Nat Rev Drug Discov.*, 3(1):935–49, 2004.

[56] R. Larson, L. Boswell, T. Kanold, and L. Stiff. *Geometry.* McDougal Littel, Evanston, IL, 2007.

[57] A. Lavecchia and C. D. Giovanni. Virtual screening strategies in drug discovery: a critical review. *Curr Med Chem*, 20(23):2839–60, 2013.

[58] R. H. Lilien, B. W. Stevens, A. C. Anderson, and B. R. Donald. A novel ensemble-based scoring and search algorithm for protein redesign, and its application to modify the substrate specificity of the gramicidin synthetase a phenylalanine adenylation enzyme. In *Proceedings of the Eighth Annual International Conference on Computational Molecular Biology, 2004, San Diego, California, USA, March 27-31, 2004*, pages 46–57, 2004.

[59] Lowrie, J.F. and et al. The different strategies for designing gpcr and kinase targeted libraries. *Comb Chem High Throughput Screen*, 7(5):495–510, 2004.

[60] I. C. Maly, D.J. and J. Ellman. Combinatorial target-guided ligand assembly: identification of potent subtype-selective c-src inhibitors. *Proc Natl Acad Sci U S A*, 97(6):2419–24, 2000.

[61] Massachusetts DOE, 2014.

[62] Mathworld.wolfram.com. Jordan curve – from wolfram mathworld, 2015.

[63] T. Matsuzaki, H. Iwane, H. Anai, and N. H. Arai. The most uncreative examinee: A first step toward wide coverage natural language math problem solving. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 1098–1104, 2014.

[64] Larson et. al. *Geometry: Practice Workbook.* McDougall Littell, Evanston, IL, 2007.

[65] Miller, J.R. and et. al. A class of selective antibacterials derived from a protein kinase inhibitor pharmacophore. *Proc Natl Acad Sci U S A*, 106(6):1737–42., 2009.

[66] Mochalkin, I. and et al. Discovery of antibacterial biotin carboxylase inhibitors by virtual screening and fragment-based approaches. *ACS Chem Biol*, 4(6):473–83, 2009.

[67] T. N. Nikolsky, Y. and A. Bugrim. Biological networks and analysis of experimental data in drug discovery. . *Drug Discov Today*, 10(9):653–62, 2005.

[68] Noel M O'Boyle and et. al. Open babel: An open chemical toolbox. *Journal of Cheminformatics*, 3(33), 2011.

[69] NY State Education Dept., 2014. `http://www.nysedregents.org/regents$_$math.html`.

[70] R. A. Orry, A.J. and C. Cavasotto. Structure-based development of target-specific compound libraries. *Drug Discov Today*, 11(5-6):261–6, 2006.

[71] R. O'Shea and H. Moser. Physicochemical properties of antibacterial compounds: implications for drug discovery. *J Med Chem*, 51(10):2871–8, 2008.

[72] A. Peleg and D. Hooper. Hospital-acquired infections due to gram-negative bacteria. *N Engl J Med.*, 362(19):1804–13., 2010.

[73] S. Renner and G. Schneider. Scaffold-hopping potential of ligand-based similarity concepts. *ChemMedChem*, 1(2):181–5, 2006.

[74] M. J. Seo, H. Hajishirzi, A. Farhadi, and O. Etzioni. Diagram understanding in geometry questions. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 2831–2838, 2014.

[75] P. Sinclair and G. Dikshit. *Mathematics Textbook for Class IX*. New Delhi, 2006. `http://www.ncert.nic.in/ncerts/textbook/textbook.htm?iemh1=0-15`.

[76] P. Sinclair and G. Dikshit. *Mathematics Textbook for Class X*. New Delhi, 2006. http://www.ncert.nic.in/ncerts/textbook/textbook.htm?jemh1=0-14.

[77] R. Singh, S. Gulwani, and S. K. Rajamani. Automatically generating algebra problems. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.*, 2012.

[78] C. T.A. *Synthesis of self-timed VLSI circuits from graph-theoretic specifications*. PhD thesis, MIT, 1987.

[79] G. Taubes. The bacteria fight back. *Science*, 321(5887):356–61., 2008.

[80] T. Vandermeersch, 2015. http://openbabel.org/wiki/Obgen.

[81] R. E. Villoutreix, B.O. and M. Miteva. Structure-based virtual ligand screening: recent success stories. *Comb Chem High Throughput Screen*, 12(10):1000–16, 2009.

[82] W. Wen-Tsün. Basic principles of mechanical theorem proving in elementary geometries. *J. Autom. Reasoning*, 2(3):221–252, 1986.

[83] Wikipedia. Protein design, 2015.

[84] L. R. o. F. Wikipedia, 2015. http://en.wikipedia.org/wiki/Lipinski%27s_rule_of_five.

[85] T. S. Wikipedia and Distance, 2015. http://en.wikipedia.org/wiki/Jaccard_index#Tanimoto_similarity_and_distance.

[86] U. C. Wikipedia, 2014. http://en.wikipedia.org/wiki/Unit_circle.

[87] S. Wilson and J. D. Fleuriot. Combining dynamic geometry, automated geometry theorem proving and diagrammatic proofs. In *Workshop on User Interfaces for Theorem Proving*, 2005.

[88] S. W. Zhang, Y.M. and C. Rock. Inhibiting bacterial fatty acid synthesis. *J Biol Chem*, 281(26):17541–4, 2006.

Vita

Chris Alvin, a native of Madison, Wisconsin, received his bachelor's degree from Ripon College in 1999. He continued his study of computer science at the University of Wisconsin at Madison acquiring a master's degree in 2001. After two years as an associate software engineer for a Department of Defense subcontractor in Laurel, Maryland, he switched careers to teach high school mathematics, statistics, and engineering. Always seeking a challenge, Chris acquired a master's degree in Mathematics from Marquette University in 2011 while a full-time teacher and in 2012 decided to commit to a doctoral degree.