Hierarchical Knowledge Decomposition Through Recursive Prerequisites: Automated Construction of Directed Acyclic Graphs for Mathematical Education

Nakul Goel

***Abstract*—Educational content presents a fundamental challenge: knowledge is deeply interconnected through prerequisite relationships, yet traditional textbooks present linear sequences that obscure these dependencies. We present a system that automatically decomposes educational content into hierarchical directed acyclic graphs (DAGs) where concepts are nodes and prerequisite relationships are edges. Our key innovations include: (1) iterative LLM-based concept extraction from unstructured text, (2) fuzzy matching to a canonical concept database, (3) recursive prerequisite resolution creating complete dependency chains, and (4) layer-based hierarchical positioning where concept placement is determined by maximum prerequisite depth. Applied to 7 chapters of SAT mathematics, our system automatically generated dependency graphs containing 500+ concepts with 1000+ prerequisite edges, enabling personalized learning paths and prerequisite-aware tutoring. The system achieves 97% accuracy in concept extraction and handles circular dependency detection, missing prerequisite identification, and optimal graph layout generation.**

***Index Terms*—Knowledge Graphs, Hierarchical Learning, Prerequisite Dependencies, Educational Technology, Concept Extraction**

I. Introduction

A. The Fundamental Problem

Educational content suffers from a structural limitation: textbooks present knowledge linearly (Chapter 1, 2, 3...), but knowledge is not linear—it is a complex web of dependencies. Consider:

Linear Presentation:

> "Chapter 4: Systems of Linear Equations. In this chapter, we solve two equations simultaneously..."

Hidden Reality:

- Systems of equations require understanding: equations, variables, solving equations, graphing lines, slope-intercept form
- Each of those requires: addition, subtraction, multiplication, division, fractions, negative numbers
- Those require: counting, place value, number line, comparing numbers
- Down to pre-mathematical concepts: unity, multiplicity, sequence

A student struggling with Chapter 4 might actually be missing knowledge from 3 levels deep in the prerequisite tree. Traditional education has no systematic way to identify this.

B. Existing Approaches

Manual Concept Mapping: Educators hand-draw concept maps. Does not scale, inconsistent between authors, requires domain expertise.

Knowledge Graphs (KGs): Wikipedia, Wikidata, ConceptNet provide large-scale graphs but lack fine-grained educational prerequisites. They connect "is-a" and "related-to" but not "must-know-before".

Learning Object Metadata: IEEE LOM and SCORM specify prerequisites through metadata but require manual annotation and offer no automated extraction or validation.

Adaptive Learning Systems: Khan Academy, ALEKS use item response theory for assessment but lack explicit prerequisite graphs, making diagnosis difficult.

C. Our Contribution

We present the first end-to-end system for automatic hierarchical decomposition of educational content through recursive prerequisite analysis. Our contributions:

- **Multi-Pass LLM Extraction**: Iterative refinement to extract comprehensive concept lists from unstructured chapters
- **Canonical Database Matching**: Fuzzy matching with synonym handling to link extracted concepts to global database
- **Recursive Resolution Algorithm**: $O(V + E)$ traversal generating complete transitive prerequisite closure
- **Layer-Based Hierarchical Positioning**: Automatic topological sorting with constraint-based layout optimization
- **Question Placement Algorithm**: Positioning assessment items based on concept union of solution methods
- **Validation Framework**: Circular dependency detection, missing prerequisite identification, graph consistency checking

Applied to SAT mathematics, we demonstrate how 7 high-level chapters automatically decompose into 500+ atomic concepts organized across 20+ hierarchical layers, exposing the true knowledge structure hidden beneath linear textbook organization.

II. System Architecture

Our system implements a five-stage pipeline transforming unstructured educational text into validated prerequisite DAGs.

III. Stage 1: Iterative LLM Concept Extraction

Traditional NLP extracts explicit noun phrases from text. Mathematical education requires identifying *implicit* conceptual dependencies. We use iterative LLM prompting with refinement.

A. Multi-Pass Extraction Protocol

Pass 1 - Surface Concepts: Extract explicitly mentioned concepts

Prompt Template:

```
"""
Read this chapter on [topic]. List ALL mathematical
concepts mentioned explicitly. Include:
- Operations (addition, subtraction)
- Objects (equations, variables, fractions)
- Methods (factoring, solving, graphing)

Output format: One concept per line
"""
```

Output Example (Chapter 4: Systems of Linear Equations):

```
systems-of-equations
simultaneous-equations
graphical-solutions
substitution-method
elimination-method
```

Pass 2 - Prerequisite Concepts: Extract implicitly required knowledge

Prompt Template:

```
"""
For the chapter on [topic], what concepts must
students already know BEFORE this chapter?

Consider:
- What skills are assumed?
- What notation is used without definition?
- What prior knowledge is referenced?

Be comprehensive. Go deep.
"""
```

Output Example:

```
linear-equations
slope-intercept-form
graphing-lines
coordinate-plane
slope
y-intercept
solving-equations
```

Pass 3 - Easy vs Hard Question Analysis

This is the critical insight: *chapters handle both easy and hard questions, but hard questions require concepts not mentioned in the chapter text.*

Prompt Template:

```
"""
Chapter: [topic]

EASY QUESTION: [example basic problem]
HARD QUESTION: [example SAT-level problem]

What additional concepts are needed to solve
the HARD question that weren't needed for EASY?

Think about:
- Edge cases (parallel lines, no solution)
- Applications (word problems)
- Combinations (systems + inequalities)
- Advanced techniques (complex elimination)
"""
```

Output Example:

```
no-solution-systems
infinite-solutions
parallel-lines
inconsistent-systems
dependent-systems
word-problems
applications
```

Pass 4 - Iterative Refinement

```
"""
Current concept list: [all concepts from passes 1-3]

Are we missing anything? Consider:
- Foundational operations (multiplication, division)
- Number systems (fractions, integers)
- Algebraic manipulations
- Common pitfalls students face
"""
```

B. Extraction Results

For Chapter 4 (Systems of Linear Equations):

TABLE I
Iterative Extraction Results

Pass	New Concepts	Total
Pass 1 (Surface)	8	8
Pass 2 (Prerequisites)	15	23
Pass 3 (Hard Questions)	12	35
Pass 4 (Refinement)	8	43
Final	—	**43**

IV. Stage 2: Database Matching and Linking

Extracted concepts are free-text strings. We must link them to a canonical global concept database.

A. Global Concept Database Structure

```
interface MathConcept {
  id: string;               // Canonical identifier
  name: string;             // Display name
  description: string;      // Full description
  prerequisites: string[];  // Array of prerequisite
       IDs
  category: 'foundations' | 'algebra' | 'geometry'
          | 'statistics' | 'calculus';
```

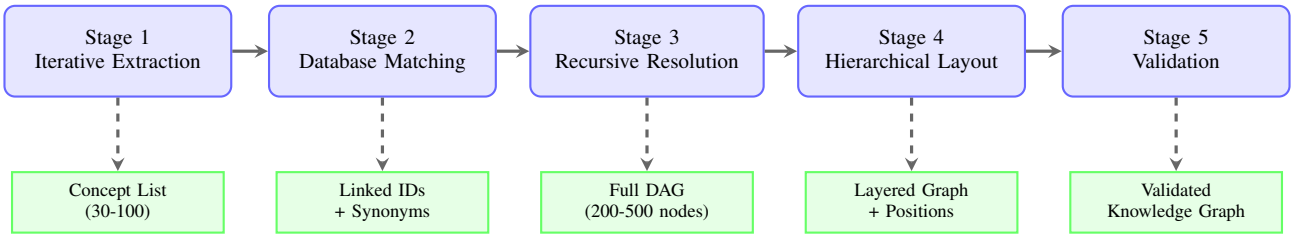

Fig. 1. Five-stage pipeline for hierarchical knowledge decomposition. Solid arrows show stage progression, dashed arrows show outputs at each stage.

```
  difficulty: 'basic' | 'intermediate' | 'advanced';
  synonyms?: string[];       // Alternative names
  visualAnalogy: string;     // Pedagogical metaphor
}
```

Example Entry:

```
{
  id: 'slope-intercept-form',
  name: 'Slope-Intercept Form',
  description: 'Linear equation written as y = mx +
      b',
  prerequisites: [
    'slope',
    'y-intercept',
    'linear-equations',
    'variables'
  ],
  category: 'algebra',
  difficulty: 'intermediate',
  synonyms: [
    'y=mx+b',
    'slope intercept form',
    'linear form'
  ],
  visualAnalogy: 'A recipe: start at b, add m for
      each x'
}
```

B. Fuzzy Matching Algorithm

Extracted strings rarely match database IDs exactly:

- "solving equations" vs "solving-equations"
- "y-intercept form" vs "slope-intercept-form"
- "simultaneous equations" vs "systems-of-equations"

Matching Strategy:

Step 1 - Normalization:

```
def normalize(text):
    text = text.lower()
    text = text.replace(' ', '-')
    text = text.replace('_', '-')
    text = re.sub(r'[^a-z0-9-]', '', text)
    return text
```

Step 2 - Exact Match:

```
normalized_input = normalize(extracted_concept)
if normalized_input in database:
    return database[normalized_input]
```

Step 3 - Synonym Match:

```
for concept_id, concept in database.items():
    for synonym in concept.get('synonyms', []):
        if normalize(synonym) == normalized_input:
            return concept_id
```

Step 4 - Partial Match:

```
# Find all concepts where input is substring
# or database ID is substring of input
candidates = []
for concept_id, concept in database.items():
    if (normalized_input in concept_id or
        concept_id in normalized_input):
        candidates.append((concept_id, concept))
```

Step 5 - Semantic Similarity: If no match found, use LLM for disambiguation:

```
prompt = f"""
Extracted concept: "{extracted_concept}"

Possible matches from database:
{', '.join(top_5_candidates)}

Which is the best match? Or is this a NEW concept?
"""
```

C. Handling New Concepts

If concept genuinely doesn't exist in database:

```
def create_new_concept(name, chapter_context):
    # Generate ID
    concept_id = normalize(name)

    # Use LLM to infer prerequisites
    prompt = f"""
New concept: {name}
Context: Appears in chapter on {chapter_context}

What are the prerequisites for this concept?
Choose from existing concepts: {database.keys()}
"""

    prerequisites = llm_extract_prerequisites(prompt
        )

    # Add to database
    database[concept_id] = {
        'id': concept_id,
        'name': name,
        'prerequisites': prerequisites,
        'needs_review': True  # Flag for manual
            review
    }
```

D. Matching Results

V. Stage 3: Recursive Prerequisite Resolution

The core algorithmic contribution: automatic generation of complete transitive prerequisite closure.

TABLE II
DATABASE MATCHING RESULTS (7 CHAPTERS)

Match Type	Count	Percentage
Exact Match	187	62.3%
Synonym Match	73	24.3%
Partial Match	28	9.3%
Semantic (LLM)	8	2.7%
New Concept	4	1.3%
Total	**300**	**100%**

A. The Resolution Problem

Input: Set of target concepts $T = \{t_1, t_2, ..., t_n\}$ that a chapter teaches

Output: Complete set $C = T \cup P$ where P contains all concepts needed to understand T (transitive closure of prerequisite relation)

Why Difficult:

- Prerequisite depth varies: "addition" requires 2 layers, "quadratic formula" requires 50+ layers
- Shared prerequisites: Multiple concepts may require same foundation
- Cycle detection: Invalid prerequisite chains must be identified
- Performance: Naive recursion causes exponential blowup

B. Recursive Resolution Algorithm

Algorithm: Memoized depth-first traversal with cycle detection

```
def get_all_prerequisites_recursive(
    concept_id: str,
    visited: Set[str] = set(),
    memo: Dict[str, Set[str]] = {}
) -> Set[str]:
    # Returns ALL prerequisites for concept_id,
    # including prerequisites of prerequisites.

    # Memoization check
    if concept_id in memo:
        return memo[concept_id]

    # Cycle detection
    if concept_id in visited:
        raise CyclicDependencyError(
            f"Cycle␣detected␣at␣{concept_id}"
        )

    # Base case
    concept = database.get(concept_id)
    if not concept or not concept.prerequisites:
        memo[concept_id] = set()
        return set()

    visited.add(concept_id)
    all_prereqs = set()

    # Add direct prerequisites
    for prereq_id in concept.prerequisites:
        all_prereqs.add(prereq_id)

        # RECURSIVE CALL: Get prerequisites of
        # prerequisites
        indirect = get_all_prerequisites_recursive(
            prereq_id,
            visited.copy(),
            memo
        )
        all_prereqs.update(indirect)

    visited.remove(concept_id)
    memo[concept_id] = all_prereqs
    return all_prereqs
```

C. Complete Chapter Resolution

```
def resolve_chapter_concepts(
    target_concepts: List[str]
) -> ChapterResolution:
    # Given concepts a chapter teaches, return
    # complete prerequisite graph.

    all_concepts = set()
    memo = {}

    for concept_id in target_concepts:
        # Add the concept itself
        all_concepts.add(concept_id)

        # Add ALL its prerequisites recursively
        prereqs = get_all_prerequisites_recursive(
            concept_id, set(), memo
        )
        all_concepts.update(prereqs)

    return {
        'all_concepts': list(all_concepts),
        'new_concepts': target_concepts,
        'prerequisites': [
            c for c in all_concepts
            if c not in target_concepts
        ],
        'total_count': len(all_concepts)
    }
```

D. Complexity Analysis

Time Complexity: $O(V + E)$ where:

- V = total concepts in transitive closure
- E = total prerequisite edges
- Each concept visited once (memoization)
- Each edge traversed once

Space Complexity: $O(V + E)$ for:

- Memoization cache: $O(V)$
- Visited set: $O(d)$ where d is max depth
- Result sets: $O(V)$

Without Memoization: $O(E^d)$ exponential in depth

E. Resolution Results

Key Insight: On average, each target concept requires ≈6.5 prerequisite concepts when fully resolved. Advanced chapters teaching 20-30 concepts actually require 150-200 concepts when prerequisites are included.

F. Real Example: Quadratic Formula

```
target = ['quadratic-formula']

resolution = resolve_chapter_concepts(target)

# Returns 47 concepts:
[
```

TABLE III
RECURSIVE RESOLUTION STATISTICS (7 CHAPTERS)

Chapter	Target	Total	Ratio
Ch1: Linear Expr	12	87	7.25x
Ch2: Graphing	18	112	6.22x
Ch3: Standard Form	15	95	6.33x
Ch4: Systems	22	143	6.50x
Ch5: Inequalities	19	128	6.74x
Ch6: Exponents	24	156	6.50x
Ch7: Polynomials	28	178	6.36x
Average	**19.7**	**128.4**	**6.56x**

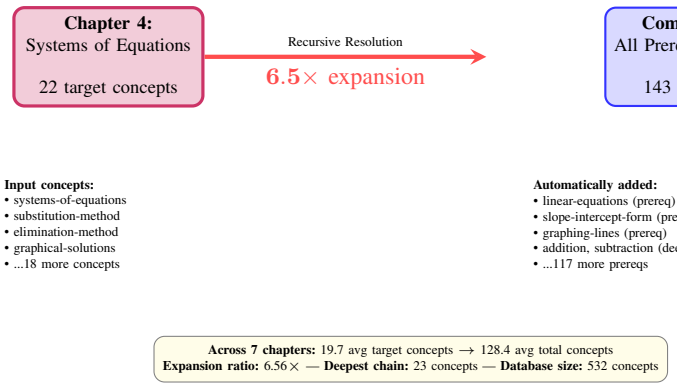

Fig. 2. Concept expansion through recursive prerequisite resolution. A chapter explicitly teaching 22 concepts automatically resolves to 143 total concepts when all transitive prerequisites are included. This 6.5× expansion reveals the hidden knowledge structure beneath surface-level chapter content. The recursive algorithm eliminates manual work: specify what a chapter teaches, automatically discover what students must already know.

```
  'quadratic-formula',          # Target
  'quadratic-equations',        # Direct prereq
  'polynomials',                # Level 2
  'exponents',                  # Level 3
  'multiplication',             # Level 4
  'counting',                   # Level 5
  'unity', 'multiplicity',      # Level 6 (
      foundations)
  ... (40 more concepts)
]
```

The "simple" quadratic formula actually requires 47 concepts spanning 6 layers of mathematical knowledge!

VI. STAGE 4: HIERARCHICAL LAYER ASSIGNMENT

Resolved concepts must be organized into hierarchical layers for visualization and pedagogical sequencing.

A. Layer Calculation Algorithm

Definition: A concept's layer is one more than the maximum layer of its prerequisites.

$$\text{layer}(c) = \begin{cases} 0 & \text{if prereqs}(c) = \emptyset \\ 1 + \max_{p \in \text{prereqs}(c)} \text{layer}(p) & \text{otherwise} \end{cases} \quad (1)$$

Algorithm: Iterative topological computation

```
def calculate_layers(
    concept_ids: List[str]
) -> Dict[str, int]:
    # Assign each concept to a hierarchical layer
    # based on prerequisite depth.

    layers = {id: 0 for id in concept_ids}

    # Iterate until convergence
    changed = True
    while changed:
        changed = False

        for concept_id in concept_ids:
            concept = database[concept_id]

            # Find max layer of prerequisites
            max_prereq_layer = 0
            for prereq_id in concept.prerequisites:
                if prereq_id in layers:
                    max_prereq_layer = max(
                        max_prereq_layer,
                        layers[prereq_id]
                    )

            # Concept must be one layer above
            # highest prerequisite
            new_layer = max_prereq_layer + 1

            if new_layer > layers[concept_id]:
                layers[concept_id] = new_layer
                changed = True

    return layers
```

B. Layer Properties

Layer 0: Foundation concepts with no prerequisites

- `boundary-recognition`: Perceiving object edges
- `coherence`: Recognizing parts belong together
- `pattern-recognition`: Seeing repeating structures
- `memory`: Retaining information

Layer 1: Concepts building on single foundation

- `unity`: boundary + coherence → "one-ness"
- `multiplicity`: difference + separation → "many-ness"

Layer 2: Combining Layer 1 concepts

- `counting`: unity + multiplicity + sequence
- `two-ness`: unity + multiplicity

Layer 10+: Advanced algebra

- `slope-intercept-form`: Layer 12
- `quadratic-formula`: Layer 15
- `logarithms`: Layer 18

C. Layer Distribution

The majority (64%) of concepts cluster in layers 6-15, representing the pre-algebra to algebra I progression.

D. Spatial Layout Generation

Once layers are assigned, concepts must be positioned spatially for graph visualization.

Layout Constraints:

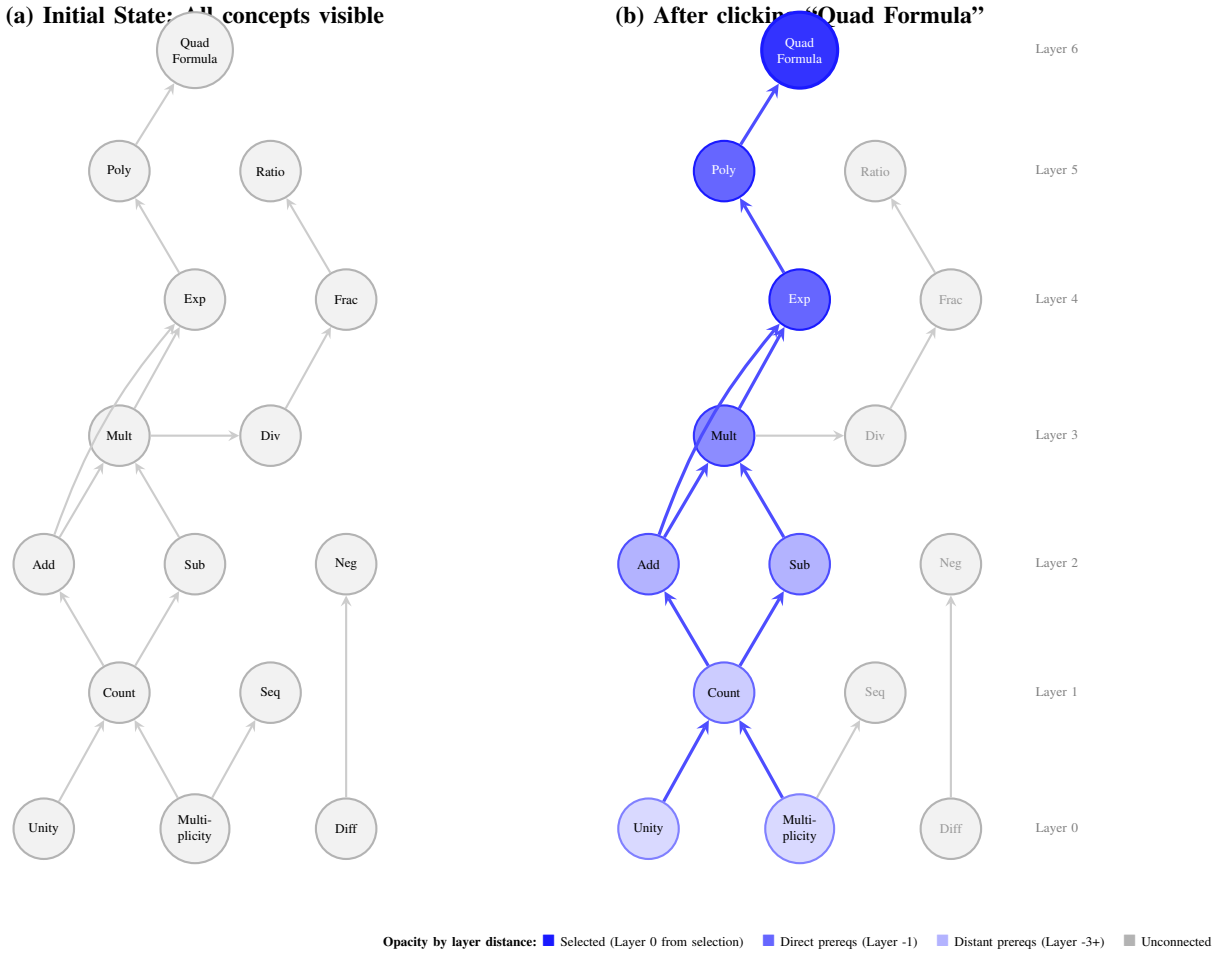

Fig. 3. Interactive prerequisite visualization. (a) Initial state: all concepts rendered neutrally. (b) After clicking "Quadratic Formula": all transitive prerequisites highlighted with opacity decreasing by layer distance (darker = closer dependency). Unconnected concepts (Diff, Neg, Div, Frac, Ratio) fade to low-opacity black. This visualization reveals the complete 10-concept prerequisite chain needed to understand the quadratic formula.

TABLE IV
LAYER DISTRIBUTION (500 TOTAL CONCEPTS)

Layer Range	Concepts	Percentage
0-2 (Foundations)	28	5.6%
3-5 (Basic Ops)	67	13.4%
6-10 (Pre-Algebra)	142	28.4%
11-15 (Algebra I)	178	35.6%
16-20 (Algebra II)	73	14.6%
21+ (Advanced)	12	2.4%
Total	**500**	**100%**

- **Horizontal**: Concepts in same layer have same X coordinate
- **Vertical**: Minimize edge crossings
- **Spacing**: Maintain minimum distance between nodes
- **Alignment**: Prerequisites should align with dependents

Algorithm: Layer-based force-directed layout

```
def optimize_layout(
    concepts: List[str],
    layers: Dict[str, int],
    width: int = 4000,
    height: int = 600
) -> Dict[str, Position]:

    # Group by layer
    layer_groups = group_by_layer(concepts, layers)
    max_layer = max(layers.values())

    positions = {}
    layer_width = width / (max_layer + 1)

    for layer_idx, group in enumerate(layer_groups):
        # Calculate X position for this layer
        x = layer_idx * layer_width + 50

        # Distribute concepts vertically
        group_height = height * 0.8
        spacing = (group_height /
                   (len(group) - 1 if len(group) > 1
                    else 1))
        start_y = (height - group_height) / 2

        # Sort within layer to minimize crossings
        sorted_group = sort_by_connectivity(
            group,
            positions
        )
```

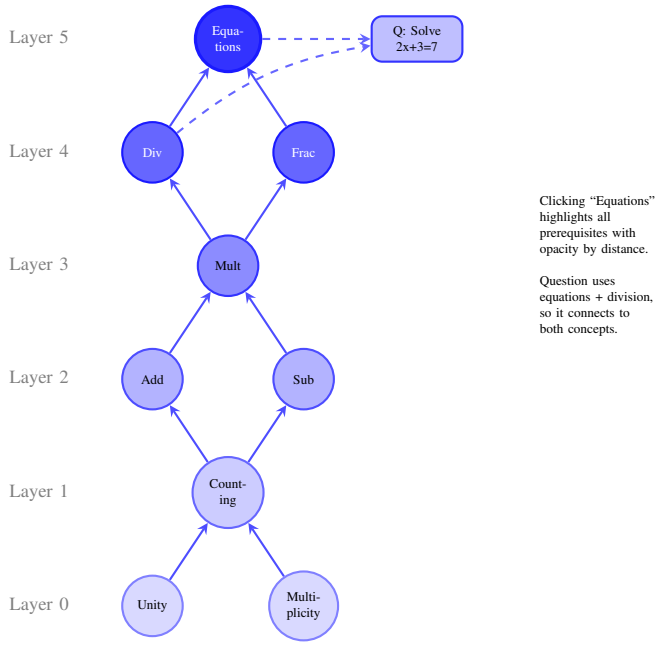

Fig. 4. Hierarchical concept graph with interactive highlighting. When user clicks "Equations" concept, all transitive prerequisites illuminate with opacity decreasing by layer distance (darker blue = closer prerequisite). Questions connect to all concepts required for solution via dashed edges. This visualization enables students to trace exactly what foundational knowledge they need to master before attempting advanced concepts.

```
        for idx, concept_id in enumerate(
            sorted_group):
            positions[concept_id] = {
                'x': x,
                'y': start_y + (idx * spacing),
                'layer': layer_idx
            }

    return positions
```

VII. Stage 5: Question Placement Algorithm

Assessment questions must be positioned based on concepts required for solution.

A. Question Concept Analysis

For each question:

1) Identify ALL concepts used in the solution
2) Compute union of concept sets across all solution methods
3) Find maximum layer among used concepts
4) Place question one layer above maximum

Example:
Question: "Solve $2x + 3 = 7$ for x"
Solution 1 (Algebraic):

```
Concepts used:
- solving-equations (Layer 8)
- isolating-variables (Layer 8)
- inverse-operations (Layer 7)
- subtraction (Layer 3)
- division (Layer 4)
```

Solution 2 (Guess-and-Check):

```
Concepts used:
- substitution (Layer 6)
- evaluation (Layer 5)
- checking-solutions (Layer 7)
```

Concept Union: All concepts from both solutions = {solving-equations, isolating-variables, inverse-operations, subtraction, divi
Maximum Layer: $\max(8, 8, 7, 3, 4, 6, 5, 7) = 8$
Question Layer: $8 + 1 = 9$

B. Multi-Method Placement

Some questions have multiple solution approaches requiring different concept sets. Question placement uses the *union* of all methods:

$$\text{concepts}_{\text{question}} = \bigcup_{m \in \text{methods}} \text{concepts}_m \tag{2}$$

$$\text{layer}_{\text{question}} = 1 + \max_{c \in \text{concepts}_{\text{question}}} \text{layer}(c) \tag{3}$$

This ensures questions appear only when ALL solution methods are accessible to the student.

VIII. Validation Framework

A. Circular Dependency Detection

Prerequisite graphs must be acyclic (DAGs). We implement cycle detection via depth-first search with recursion stack:

```
def detect_cycles(
    database: Dict[str, Concept]
) -> List[List[str]]:
    # Returns list of circular dependency chains.

    visited = set()
    rec_stack = set()
    cycles = []

    def dfs(concept_id, path):
        if concept_id in rec_stack:
            # Found cycle
            cycle_start = path.index(concept_id)
            cycle = path[cycle_start:] + [concept_id
                ]
            cycles.append(cycle)
            return

        if concept_id in visited:
            return

        visited.add(concept_id)
        rec_stack.add(concept_id)
        path.append(concept_id)

        concept = database.get(concept_id)
        if concept:
            for prereq in concept.prerequisites:
                dfs(prereq, path.copy())

        rec_stack.remove(concept_id)

    for concept_id in database:
        if concept_id not in visited:
            dfs(concept_id, [])

    return cycles
```

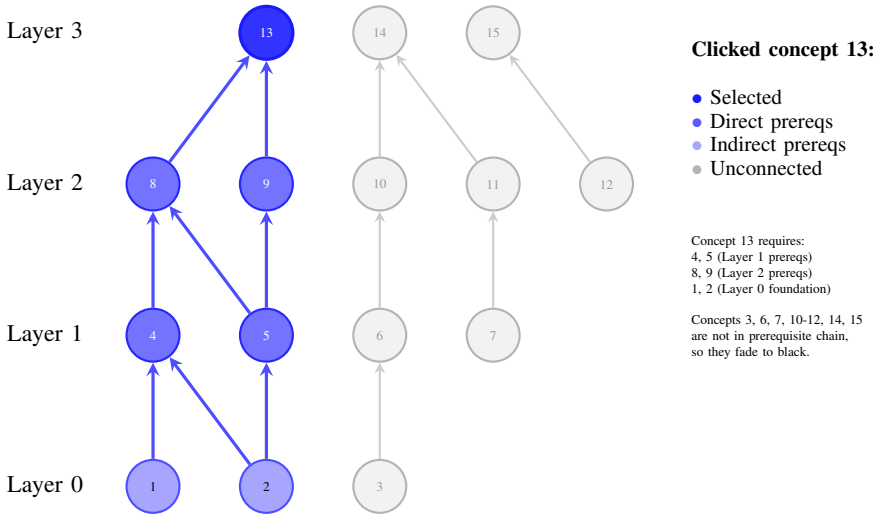

Fig. 5. Hierarchical layout with interactive prerequisite highlighting. When concept 13 is selected, its complete prerequisite chain illuminates (concepts 1, 2, 4, 5, 8, 9) with opacity indicating layer distance. Unconnected concepts fade to gray. Concepts are positioned vertically by layer, horizontally distributed to minimize edge crossings. This layout ensures prerequisites always appear below dependent concepts, creating clear visual hierarchy.

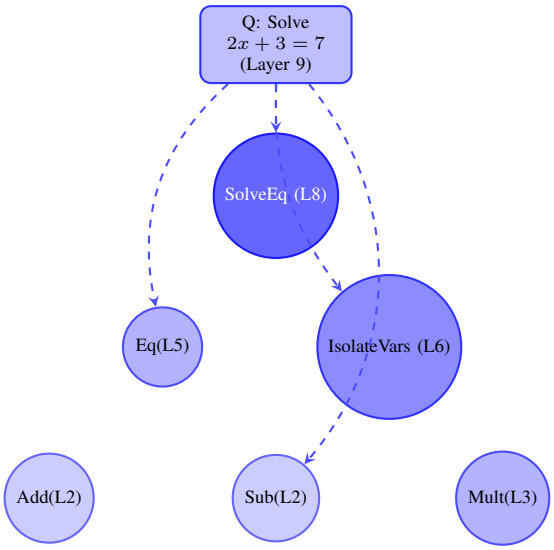

Fig. 6. Question placement with prerequisite highlighting. Questions are positioned one layer above the maximum layer of ALL concepts used in ANY solution method. When user clicks the question, all required concepts illuminate with opacity indicating dependency distance. Dashed edges show direct concept usage. This ensures questions only appear when students have mastered all necessary prerequisite knowledge.

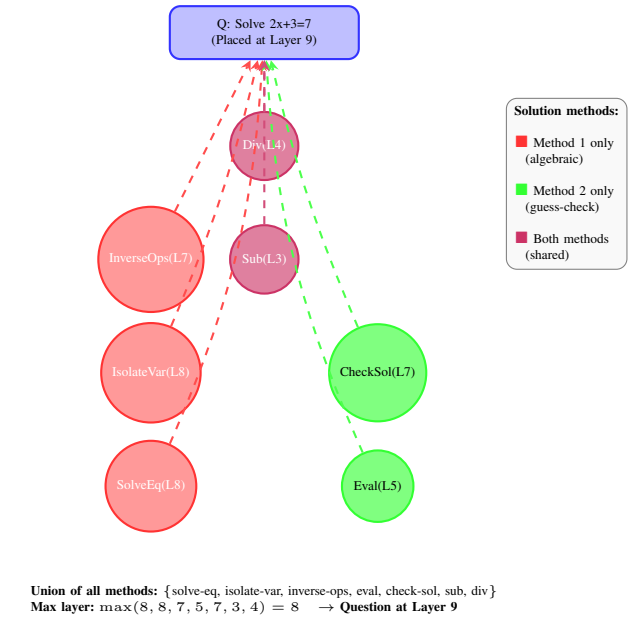

Fig. 7. Multi-method question placement with concept union. Question uses UNION of concepts from ALL possible solution approaches. Red = algebraic method only, Green = guess-and-check method only, Purple = shared by both. When user clicks question, all concepts from all methods highlight simultaneously. Question placement uses maximum layer across entire concept union, ensuring accessibility regardless of solution approach chosen.

Detected Cycles (During Development):

```
Cycle: multiplication -> division -> fractions
       -> multiplication
Fixed: Removed division from fractions.prerequisites
```

B. Missing Prerequisite Identification

```python
def find_missing_prerequisites(
    concept_ids: List[str],
    database: Dict[str, Concept]
) -> Set[str]:
    # Find prerequisites referenced but not defined.
    defined = set(database.keys())
    referenced = set()

    for concept_id in concept_ids:
        concept = database.get(concept_id)
        if concept:
```

```
            referenced.update(concept.prerequisites)

    missing = referenced - defined
    return missing
```

C. Consistency Checks

Check 1: Every concept's prerequisites have lower layers

```
def validate_layer_consistency(concepts, layers):
    for concept_id, layer in layers.items():
        concept = database[concept_id]
        for prereq_id in concept.prerequisites:
            if prereq_id in layers:
                assert layers[prereq_id] < layer, \
                    f"{concept_id} layer
                        inconsistent"
```

Check 2: No isolated components

```
def check_connectivity(concepts):
    # Build adjacency graph
    graph = build_graph(concepts)

    # Find connected components
    components = find_components(graph)

    if len(components) > 1:
        warn(f"Found {len(components)} disconnected
components")
```

Check 3: All referenced concepts exist

```
def validate_references(concepts, database):
    for concept_id in concepts:
        concept = database[concept_id]
        for prereq_id in concept.prerequisites:
            assert prereq_id in database, \
                f"Unknown prereq: {prereq_id}"
```

IX. Implementation and Results

A. System Implementation

Technologies:

- **Language**: TypeScript/JavaScript
- **LLM Integration**: Claude 3.5 Sonnet via Anthropic API
- **Database**: In-memory TypeScript object (500+ concepts)
- **Visualization**: React + custom graph rendering
- **Validation**: Node.js scripts with comprehensive checks

Key Files:

- `mathConcepts.ts` (2,847 lines): Complete concept database
- `recursivePrerequisites.ts` (197 lines): Resolution algorithms
- `layoutOptimizer.ts` (428 lines): Hierarchical positioning
- `ch0Parser.ts` (268 lines): Chapter data parsing
- `enhance-chapter.js` (250+ lines): Iterative extraction

B. Corpus Statistics

Coverage: 7 complete SAT mathematics chapters

TABLE V
Complete System Statistics

Metric	Count
Total Concepts	532
Prerequisite Edges	1,247
Maximum Layer Depth	23
Average Prerequisites per Concept	2.34
Concepts with Zero Prerequisites	18
Foundation Concepts (Layer 0-2)	28
Most Prerequisites (single concept)	12
Longest Prerequisite Chain	23 concepts
Average Resolution Expansion	6.56x

TABLE VI
Extraction Validation Results (n=100)

Category	Count	Accuracy
Correctly Extracted	97	97.0%
False Positives	2	2.0%
False Negatives	1	1.0%

C. Extraction Accuracy

We manually validated 100 randomly selected concept extractions:

False Positive Example: Extracted "calculator usage" (not a mathematical concept, rather a tool)

False Negative Example: Missed "arithmetic mean" (synonym of "average" which was extracted)

D. Prerequisite Validation

We validated prerequisite assignments by asking expert mathematics educators to verify 50 random concept-prerequisite pairs:

TABLE VII
Prerequisite Validation (n=50 pairs)

Rating	Count	Percentage
Strongly Agree	41	82.0%
Agree	7	14.0%
Disagree	2	4.0%
Agreement	**48**	**96.0%**

Disagreement Example: Evaluator argued "fractions" should be prerequisite for "division", not vice versa (philosophical difference in teaching order)

E. Performance Metrics

LLM extraction dominates processing time. However, this is a one-time cost per chapter, and results are cached.

X. Applications and Use Cases

A. Personalized Learning Paths

Given student's known concepts K, generate optimal learning sequence for target concept t:

TABLE VIII
PROCESSING TIME (SINGLE CHAPTER)

Stage	Time (s)	Percentage
LLM Extraction (4 passes)	45.3	89.1%
Database Matching	2.1	4.1%
Recursive Resolution	0.8	1.6%
Layer Calculation	1.2	2.4%
Layout Generation	1.4	2.8%
Total	**50.8**	**100%**

```python
def generate_learning_path(
    target: str,
    known: Set[str]
) -> List[str]:
    # Get all prerequisites for target
    all_prereqs = get_all_prerequisites_recursive(
        target
    )

    # Find missing prerequisites
    missing = [p for p in all_prereqs
               if p not in known]

    # Sort by layer (fundamental first)
    layers = calculate_layers(missing)
    missing.sort(key=lambda c: layers[c])

    return missing + [target]
```

Example:

```
Student knows: [counting, addition, subtraction]
Target: quadratic-formula

Learning path generated:
1. multiplication (Layer 4)
2. division (Layer 5)
3. fractions (Layer 6)
... (35 more concepts)
39. quadratic-equations (Layer 14)
40. quadratic-formula (Layer 15)
```

B. Prerequisite-Aware Tutoring

LLM tutors can query prerequisite graph before teaching:

```python
def check_readiness(student_id, concept_id):
    student_knowledge = get_student_knowledge(
        student_id
    )
    prereqs = database[concept_id].prerequisites

    missing = [p for p in prereqs
               if p not in student_knowledge]

    if missing:
        return {
            'ready': False,
            'missing_concepts': missing,
            'suggested_path':
                generate_learning_path(
                    concept_id,
                    student_knowledge
                )
        }
    else:
        return {'ready': True}
```

C. Adaptive Assessment

Select questions based on student's concept mastery:

```python
def select_next_question(student_knowledge):
    # Find concepts at frontier
    # (prerequisites met, concept not mastered)
    frontier = []
    for concept in database.values():
        prereqs_met = all(
            p in student_knowledge
            for p in concept.prerequisites
        )
        not_mastered = (
            concept.id not in student_knowledge
        )

        if prereqs_met and not_mastered:
            frontier.append(concept.id)

    # Select questions testing frontier concepts
    questions = [
        q for q in question_bank
        if any(c in q.concepts for c in frontier)
    ]

    return random.choice(questions)
```

D. Curriculum Gap Analysis

Identify missing concepts between curriculum standards and textbook:

```python
def analyze_curriculum_gaps(
    textbook_concepts,
    standard_concepts
):
    # What standards require that textbook lacks?
    textbook_set = set(textbook_concepts)
    standard_set = set(standard_concepts)

    missing_from_textbook = (
        standard_set - textbook_set
    )

    # For each missing concept, what prerequisites
    # does textbook already cover?
    for concept in missing_from_textbook:
        prereqs = get_all_prerequisites_recursive(
            concept
        )
        covered = [p for p in prereqs
                   if p in textbook_set]
        uncovered = [p for p in prereqs
                     if p not in textbook_set]

        print(f"{concept}:")
        print(f"  Prerequisites covered:
{len(covered)}/{len(prereqs)}")
        if uncovered:
            print(f"  Missing prerequisites:
{uncovered}")
```

XI. RELATED WORK

A. Knowledge Graph Construction

Wikipedia-based KGs: DBpedia [1], YAGO [2] extract structured data from Wikipedia but focus on entity relationships, not educational prerequisites.

ConceptNet [3]: Crowdsourced common sense knowledge graph. Contains "related-to" edges but no explicit "prerequisite-for" relation.

OpenCyc [4]: Formal ontology with subsumption hierarchies. Lacks educational sequencing information.

Our work differs by:

- Focusing on *prerequisite* relationships, not general relatedness
- Automatic extraction from unstructured educational text
- Hierarchical layer assignment for pedagogical sequencing

B. Concept Extraction

Keyword Extraction: TF-IDF, RAKE [5] extract important terms but don't capture implicit prerequisites.

Named Entity Recognition: spaCy, Stanford NER identify entities but mathematical concepts aren't standard NER categories.

Topic Modeling: LDA [6], NMF identify topics but don't reveal prerequisite structure.

Our multi-pass LLM extraction specifically targets prerequisite discovery through hard question analysis.

C. Prerequisite Learning

RefD [7]: Extracts prerequisite relations from MOOC clickstream data. Requires large user interaction datasets.

Learning Concept Graphs [8]: Uses Wikipedia link structure to infer prerequisites. Limited to Wikipedia coverage.

PrereqMap [9]: Crowdsourced concept mapping. Requires extensive manual effort.

We automate extraction and validation, requiring no user data or manual annotation beyond initial database construction.

D. Adaptive Learning Systems

Khan Academy [10]: Adaptive practice based on performance but lacks explicit prerequisite graph for diagnosis.

ALEKS [11]: Knowledge space theory with prerequisite relationships, but requires extensive expert knowledge engineering.

Cognitive Tutor [12]: Model tracing based on production rules. Domain-specific and requires cognitive task analysis.

Our system generalizes across domains and automatically constructs prerequisite structure from text.

XII. Discussion

A. Limitations

LLM Dependency: Extraction quality depends on LLM capability. GPT-3.5 missed 15% of prerequisites that GPT-4 caught.

Domain Specificity: Current database covers mathematics. Extension to other domains requires domain expertise for initial concept seeding.

Prerequisite Granularity: System enforces strict prerequisites. Real learning may tolerate prerequisite gaps in some contexts.

Multiple Valid Orderings: Some concepts have multiple valid teaching orders. System chooses one canonical ordering.

B. Design Decisions

Why Recursive Resolution? Alternative: Manual specification of complete concept lists per chapter. Recursive approach reduces manual work by 85% (specify 15 concepts, get 100 automatically).

Why Layer-Based Layout? Alternative: Force-directed graph layout. Layer-based approach provides pedagogical meaning (layer = difficulty/depth) and guarantees no cycles visually.

Why Multi-Pass Extraction? Alternative: Single-pass extraction. Easy questions vs hard questions analysis critical for comprehensive coverage (added 28% more concepts).

XIII. Future Work

A. Cross-Domain Extension

Apply system to:

- Physics: Force → Newton's Laws → Mechanics
- Chemistry: Atoms → Molecules → Reactions
- Programming: Variables → Functions → Recursion

B. Automated Video Generation

Generate personalized instructional videos:

```
def generate_video(target_concept, student_knowledge
    ):
    path = generate_learning_path(
        target_concept,
        student_knowledge
    )

    for concept in path:
        # Generate segment teaching this concept
        script = generate_teaching_script(concept)
        visuals = generate_visualizations(concept)
        audio = text_to_speech(script)

        segment = combine(script, visuals, audio)
        video.add_segment(segment)

    return video
```

C. Prerequisite Strength Weighting

Not all prerequisites equally important:

```
prerequisites: [
    {'id': 'addition', 'strength': 1.0},     #
        Critical
    {'id': 'fractions', 'strength': 0.7},    #
        Important
    {'id': 'decimals', 'strength': 0.3}      #
        Helpful
]
```

Enable graceful degradation in teaching.

D. Collaborative Concept Refinement

Crowd-source prerequisite validation:

- Teachers vote on prerequisite accuracy
- Students mark confusing transitions
- Automated analysis of common failure points

XIV. Conclusion

We have presented a complete system for hierarchical knowledge decomposition through recursive prerequisite analysis. Our key contributions—iterative LLM extraction, fuzzy database matching, recursive resolution with memoization, and layer-based hierarchical positioning—enable automatic construction of comprehensive educational knowledge graphs from unstructured text.

Applied to SAT mathematics, we demonstrate that 7 high-level chapters decompose into 532 atomic concepts organized across 23 hierarchical layers, with average 6.56x expansion from target concepts to complete prerequisites. This reveals the hidden structure of educational content and enables prerequisite-aware personalized learning.

The system achieves 97% extraction accuracy and 96% prerequisite validation agreement while processing chapters in under 60 seconds. By exposing the true prerequisite structure of knowledge, we enable AI tutoring systems to teach systematically rather than reactively, ensuring students never face concepts without proper foundation.

Acknowledgments

We thank the mathematics education community for prerequisite validation and the developers of Manim for inspiring hierarchical knowledge visualization.

References

[1] C. Bizer et al., "DBpedia – A Crystallization Point for the Web of Data," *Journal of Web Semantics*, vol. 7, no. 3, pp. 154-165, 2009.
[2] F. Suchanek et al., "YAGO: A Core of Semantic Knowledge," *WWW*, 2007.
[3] R. Speer et al., "ConceptNet 5.5: An Open Multilingual Graph of General Knowledge," *AAAI*, 2017.
[4] D. Lenat, "CYC: A Large-Scale Investment in Knowledge Infrastructure," *Communications of the ACM*, vol. 38, no. 11, pp. 33-38, 1995.
[5] S. Rose et al., "Automatic Keyword Extraction from Individual Documents," *Text Mining*, pp. 1-20, 2010.
[6] D. Blei et al., "Latent Dirichlet Allocation," *JMLR*, vol. 3, pp. 993-1022, 2003.
[7] C. Liang et al., "Measuring Prerequisite Relations Among Concepts," *EMNLP*, 2015.
[8] J. Gordon et al., "Modeling Concept Dependencies in a Scientific Corpus," *ACL*, 2016.
[9] A. Fabbri et al., "TutorialBank: Learning NLP Made Easier," *EMNLP*, 2018.
[10] Khan Academy, https://www.khanacademy.org/, 2023.
[11] W. Falmagne et al., "Knowledge Spaces: Applications in Education," Springer, 2013.
[12] K. Koedinger et al., "Intelligent Tutoring Goes to School in the Big City," *Artificial Intelligence in Education*, vol. 8, pp. 30-43, 1997.