Knowledge Graph Collector
採用理論と仕組みの技術解説書

Wikipedia、Google
Trends、TF-IDF、グラフ理論、Ollama/Gemma、Web可視化の関係を一冊で読む

このPDFは、現在開発中のナレッジグラフ自動生成システムで使われている理論やアルゴリズムを、実装との対応がわかるように整理したものです。対象読者は、Pythonは読めるが情報検索やグラフ理論はこれから学ぶ人を想定しています。

1. システム全体の考え方

このシステムは、学びたいトピックを入力すると、そのトピックを理解するために必要な概念をノード、概念同士の前提関係をエッジとして表現します。出発点はWikipediaのリンクと本文です。リンクは候補語を広く拾うために使い、本文は候補が本当にトピックに近いかを判断するために使います。Google Trendsは検索量を測り、難易度や注目度の補助信号として使います。さらに現在の主経路では、Ollama/Gemmaが教育設計者として候補を整理し、学習順序に近いグラフへ変換します。

要素	このシステムでの役割
Wikipediaリンク	関連候補を広く集める。ページ間リンクから関係候補も得る。
Wikipedia本文	TF-IDF類似度、ページ分類、候補フィルタに使う。
Google Trends	検索ボリュームを取得し、難易度スコアの補助にする。
NetworkX	有向グラフ、循環検出、トポロジカルソートに使う。
Pydantic	ノード、エッジ、グラフのJSON構造を検証・正規化する。
Ollama/Gemma	教育的に妥当なノード選択、layer分類、依存エッジ作成を補助する。
Web Canvas	生成されたグラフをブラウザ上に描画する。

処理の大まかな流れは次の通りです。

```
入力トピック -> 調査計画 -> Wikipedia候補収集 -> 候補フィルタ -> Trends取得 -> グラフ生成 -> JSON保存 -> Web表示
```

2. Wikipediaリンク構造を使う理由

Wikipediaページには本文だけでなく、多数の内部リンクがあります。内部リンクは、そのページを説明するために必要または関連がある概念への参照です。たとえば「機械学習」のページには、教師あり学習、ニューラルネットワーク、統計学、最適化、人工知能などへのリンクが含まれます。この性質を利用すると、LLMなしでも関連概念の候補集合を作れます。

・ リンクは網羅性が高い。人間が執筆した百科事典の参照構造なので、候補の取りこぼしを減らせる。

・ リンクはノイズも多い。年号、人物、企業、識別子、場所も混ざるため、そのまま学習ノードにはできない。

・ そこで本文量、被リンク数、TF-IDF類似度、ページタイプ分類を使って候補を削る。

実装上はMediaWiki APIの action=query, prop=links|extracts
を使い、リンク一覧と本文抽出を同時に取得します。HTTP
429が返った場合は待機して再試行し、ページ取得結果は簡易キャッシュに保存します。

3. トークン化

TF-IDFや類似度計算の最初のステップは、文章を単語のような単位に分けることです。このシステムでは形態素解析器を使わず、正規表現で軽量に分割しています。

```
re.findall(r"[a-z0-9]+|[一-龠々仝一]+|[ぁ-□]+|[ァ-ヶ]+", normalized)
```

対象	拾う文字列の例
英数字	python, api, 2026
漢字	機械学習, 統計, 関数
ひらがな	とは, する
カタカナ	データ, モデル

この方式はMeCabやSudachiのような本格的な日本語形態素解析より粗いですが、依存ライブラリが少なく、Wikipedia候補をざっくり絞る用途には十分軽量です。

4. TF-IDF類似度

TF-IDFは、文書内によく出る語を重視しつつ、どの文書にも出る一般語の価値を下げるための古典的な情報検索手法です。このシステムでは、トピック本文と候補ページ本文の近さを測るために使います。

TF: Term Frequency。ある文章内で語が何回出るか。

IDF: Inverse Document Frequency。珍しい語ほど大きくする重み。

実装では一般的な log(N / df) ではなく、簡易式 idf = doc_count / doc_freq[term]
を使っています。比較対象が2文書なら、両方に出る語は1、片方だけの語は2になります。

例:
A = Python は プログラミング 言語 です
B = Python は データ分析 に 使われる プログラミング 言語 です

語	AでのTF	BでのTF	df	idf
Python	1	1	2	1.0
プログラミング	1	1	2	1.0
言語	1	1	2	1.0
データ分析	0	1	1	2.0
使われる	0	1	1	2.0

Aのベクトルは共通語だけを持ち、Bのベクトルは共通語に加えてデータ分析などの固有語を持ちます。最後に2つのTF-IDFベクトルのコサイン類似度を計算します。

```
cosine_similarity = dot(vector_a, vector_b) / (norm(vector_a) * norm(vector_b))
```

この値が高いほど、候補ページは元トピックページと語彙的に近いと判断されます。filter_skillsでは similarity < 0.05 の候補を除外します。

5. コサイン類似度

コサイン類似度は、2つのベクトルの向きの近さを測ります。文書の長さそのものではなく、語の分布が似ているかを見るため、長いWikipedia本文と短い本文を比較するときにも使いやすい指標です。

・ 1.0に近い: 使われている語の方向がかなり近い。

・ 0.0に近い: 共通する語が少ない、または分布がまったく違う。

・ このシステムでは負の値は基本的に出ない。語の出現回数ベースだから。

注意点として、TF-IDF類似度は意味を直接理解しているわけではありません。「ニューラルネットワーク」と「深層学習」は意味的に近くても、本文中の語が重ならなければ低く出ることがあります。そのため、このシステムではTF-IDFを最終判断ではなく、候補を掃除するための一次フィルタとして使っています。

6. 被リンク数による重要度

候補ページ同士のリンクマップを作ると、ある候補が他の候補から何回参照されているかを数えられます。これが簡易的な被リンク数です。多くの関連候補から参照されるページは、トピック周辺で中心的な概念である可能性が高いと考えます。

```
例:
教師あり学習 -> 損失関数, 回帰分析
ニューラルネットワーク -> 損失関数, 最適化
回帰分析 -> 損失関数
この場合、損失関数は3回参照されるため重要候補になりやすい。
```

filter_skillsでは min_backlinks を使い、被リンクが少なすぎる候補を除外します。これはWikipediaリンク由来の偶然のノイズを減らす役割を持ちます。

7. ページタイプ分類

classify_page_typeは、ページタイトルと本文に含まれるキーワードからページの大まかな種類を推定します。機械学習ではなくルールベースです。

分類	判定に使う語の例	用途
technology	プログラミング, API, software, library	技術・概念として残しやすい
place	都市, 県, 国, 島, 駅	学習ノードとしては除外候補
organization	企業, 会社, 大学, 協会	文脈情報として扱いやすい
person	人物, 作家, 選手, 皇帝	学習概念ではないことが多い
event	戦争, 事件, 会議, 年, 時代	歴史的背景として扱う
concept	上記以外	一般概念

この分類は get_semantic_enrichment で relationship_hint を作るときにも使われます。たとえば技術同士なら prerequisite、人物や場所なら context といった関係ヒントになります。

8. Google Trendsと検索ボリューム

Google Trendsは、検索語がどれくらい注目されているかを0から100程度の相対値として返します。このシステムでは、候補語を5件ずつまとめて取得し、過去12か月の平均値を検索ボリュームとして扱います。

```
pytrends.build_payload(chunk, timeframe="today 12-m", geo="JP")
data = pytrends.interest_over_time()
volume = mean(data[skill])
```

検索ボリュームは「多くの人が調べている語は学習上つまずきやすい、または重要度が高いかもしれない」という補助仮説に基づきます。ただし検索量は難しさそのものではありません。流行、ニュース、固有名詞の影響を受けるため、難易度スコアの一部として控えめに使うのが安全です。

9. 正規化とdifficulty_score

非AIの graph_builder.py では、検索ボリュームを0から1に正規化して difficulty_score にします。最大検索量を持つ語が1.0、検索量0の語が0.0です。

```
difficulty_score(skill) = search_volume(skill) / max(search_volume)
```

候補	検索量	正規化後
Python	100	1.00
関数	60	0.60
ジェネレータ	20	0.20
特殊メソッド	0	0.00

全候補の検索量が0の場合は、ゼロ除算を避けるため全ノードの difficulty_score を0.0にします。

10. グラフ理論: ノードとエッジ

ナレッジグラフは、概念をノード、概念間の関係をエッジとして表した構造です。このシステムでは、学習前提を表す有向エッジを使います。

```
A -> B は「Aを理解してからBを学ぶとよい」または「AがBの前提である」という意味です。
```

非AIモードでは、WikipediaのAページがBにリンクしている場合、「BはAを説明するために参照されているので、BはAの前提知識かもしれない」と解釈し、B -> A の向きに反転してエッジを作ります。

```
例:
ページ「関数」が「変数」にリンクしている
=> 変数 -> 関数
=> 変数を知ってから関数を学ぶ
```

11. 有向非巡回グラフとトポロジカルソート

学習順序を決めるには、前提関係に循環がないことが望ましいです。AがBの前提で、BがCの前提で、CがAの前提だと、どこから学べばよいかわかりません。循環のない有向グラフをDAG、Directed Acyclic Graphと呼びます。

assign_layersではNetworkXでDiGraphを作り、循環が見つかったら循環内のエッジを1本ずつ削除します。その後、topological_sortで前提から順に並べます。

```
layer(node) = 0 if predecessors is empty
layer(node) = max(layer(pred) + 1) otherwise
```

この計算により、前提ノードがないものはlayer 0、そこから派生する概念はlayer 1以降になります。ただし現在のGemma主パイプラインでは、教育設計ルールによりlayerを0から3に制限してLLMが直接割り当てます。

12. 循環依存の除去

Wikipediaリンクは百科事典的な相互参照なので、循環が頻繁に発生します。たとえば「Python」が「プログラミング言語」にリンクし、「プログラミング言語」も「Python」にリンクすることがあります。これは百科事典としては自然ですが、学習順序としては循環です。

このシステムでは、NetworkXの find_cycle で循環を見つけ、循環を構成するエッジの一部を削除します。完全に正しい依存関係を保証する処理ではありませんが、トポロジカルソートを可能にするための実用的な処理です。

状態	処理
循環なし	そのままlayer計算
循環あり	循環エッジを1本削除して再検査
NetworkXなし	簡易フォールバックでlayerを伝播計算

13. Pydanticによるスキーマ検証

Pydanticは、Pythonのデータを型付きモデルとして検証するライブラリです。LLMや外部APIの出力は揺れやすいため、保存前に構造を整える必要があります。

モデル	主なフィールド
KnowledgeNode	node_id, label, type, layer, description, prerequisites, mastery, difficulty_score
KnowledgeEdge	edge_id, source_id, target_id, relationship, weight, description
KnowledgeGraph	graph_id, topic, created_at, updated_at, nodes, edges, meta

difficulty_scoreやweightは0から1に制限されます。layerも0から3に正規化されます。これにより、後段のWeb表示や学習管理機能が、壊れたJSONに引きずられにくくなります。

14. Ollama/Gemmaによる教育設計

現在の主パイプラインでは、WikipediaとTrendsで集めた候補をそのままグラフにせず、Ollama上のGemmaモデルに教育設計を依頼します。これは、Wikipediaリンクが百科事典的であり、必ずしも学習順序を表さないためです。

・ 調査計画: 入力文を読み、Wikipediaで調べるseed_termsを作る。

・ 候補収集: seed_termsごとにWikipedia候補を集める。

・ 教育グラフ生成: 候補の中から学習概念だけを選び、layer 0から3と依存エッジを作る。

・ JSON強制: 出力はJSONのみとし、解析失敗時は再試行する。

重要なのは、LLMは外部情報を直接検索する役ではなく、WikipediaとTrendsから得た候補を教育的に整理する役だという点です。情報源と構成役を分けることで、候補の根拠を比較的追いやすくしています。

15. プロンプト制約とJSONパース

LLM出力は自由文になりがちです。そのためgemma_graph_builder.pyでは、JSONのみを返すよう強く指示し、さらにコード側でもJSON抽出と再試行を行います。

```
1. ```json fenced block があれば中身を取り出す
2. なければ最初の { から最後の } までを抽出
3. json.loads で辞書としてパース
4. 失敗したら「JSONオブジェクトのみ」と再指示して再試行
```

この仕組みは、LLMを使うシステムでよく使われるガードレールです。自然文を生成するモデルに対し、プログラムで扱える構造化データを返させるための実用的な防御策です。

16. layer 0から3の教育設計

layer	意味	例: 機械学習
0	前提知識	線形代数, 確率, 微分
1	基礎概念	データセット, 特徴量, 損失関数
2	中核概念	教師あり学習, 回帰, 分類
3	応用・発展	深層学習, 強化学習, モデル評価

非AIモードのlayerはグラフ構造から機械的に計算されます。一方、Gemma主モードのlayerは教育上の意味を持つ4段階です。つまり、同じlayerという名前でも、モードによって由来が異なります。

17. 理解度テストと弱点ノード

ollama_agent.pyには、グラフから理解度テストを作り、回答を採点して弱点ノードを見つける仕組みがあります。これは学習支援システムとして、単にグラフを作るだけでなく、学習者の状態に応じてグラフを更新するための準備です。

・ 問題対象ノードはlayerとdifficulty_scoreで選ばれる。

・ 各問題にはtarget_nodeとrequired_nodesが付く。

・ 採点は期待キーワードが回答に含まれるかを見る簡易方式。

・ 間違えた場合、target_nodeとrequired_nodesが弱点ノードとして記録される。

・ 弱点ノードはdifficulty_scoreが少し上がり、優先復習候補になる。

現在の採点は意味理解ではなくキーワード一致です。将来的には、埋め込み類似度やルーブリック評価を使うとより自然になります。

18. Web可視化の仕組み

web_app.pyは、Python標準ライブラリのThreadingHTTPServerを使った軽量サーバーです。フロントエンドはHTML、CSS、JavaScriptを1つの文字列として返し、Canvasでグラフを描画します。

API	役割
GET /	可視化画面HTMLを返す
GET /graph-files	output/graphs配下のJSON一覧を返す
GET /graph.json	最新または指定されたグラフJSONを返す
POST /generate	入力topicから新しいグラフを生成する

Canvas上では、layerごとに横位置を変え、同じlayer内のノードを縦に並べます。エッジはsource_id/target_idまたはfrom/toの両形式に対応しています。ノードをクリックすると、前提ノード、次に学ぶノード、説明、masteryなどの詳細が表示されます。

19. ファイル安全性

Webアプリでは、過去に生成したJSONを選択表示できます。このとき任意のファイルパスを読めてしまうと危険なので、_safe_relative_pathでプロジェクト配下のファイルだけに制限しています。

1. 指定パスが絶対パスならPROJECT_ROOTからの相対パスに変換
2. PROJECT_ROOT / candidate をresolve
3. resolvedがPROJECT_ROOT配下でなければ拒否
4. ファイルが存在する場合だけ読む

20. リトライとレート制限対策

外部サービスを使うシステムでは、ネットワーク失敗やレート制限を前提にする必要があります。このシステムは、WikipediaとGoogle Trendsの両方で待機と再試行を入れています。

対象	対策
Wikipedia	HTTP 429を検知し、Retry-Afterまたは段階的待機で再試行
Google Trends	5件ずつ取得、失敗時最大3回、チャンクごとにsleep
Ollama	JSON解析失敗時に再プロンプト、最大3回
Web送信	BrokenPipeErrorなどを握り、クライアント切断として処理

21. このシステムで各理論がどう役立つか

理論・仕組み	役立つ場面	限界
Wikipediaリンクグラフ	候補概念を広く集める	ノイズが多い
TF-IDF	トピック本文と候補本文の近さを測る	意味理解ではない
コサイン類似度	文書長に左右されにくく語彙分布を比較	語彙が違う同義語に弱い
被リンク数	中心的な候補を見つける	人気ページや一般語に偏る
ページタイプ分類	人物・場所・組織などを文脈扱いにする	ルールベースなので誤分類あり
Google Trends	注目度や難易度の補助信号	流行やニュースに影響される
DAG	学習順序を破綻させない	循環削除の妥当性は保証しない
トポロジカルソート	前提から順にlayerを決める	エッジ品質に依存する
Pydantic	JSON構造を安定させる	意味的正しさは保証しない
Ollama/Gemma	教育的なノード選択と依存設計	モデル品質とプロンプトに依存する

22. 改善の方向性

- 非AIモードとLLM補助モードの入口を明確に分ける。
- 日本語形態素解析を導入し、TF-IDFの語分割品質を上げる。
- IDFを log((N + 1) / (df + 1)) + 1 のような標準形に近づける。
- Wikipediaリンクだけでなくカテゴリ、冒頭定義文、セクション構造を使う。
- Google Trendsの検索量を難易度ではなく注目度として別フィールド化する。
- 生成処理を非同期ジョブにして、Web画面に進捗を細かく表示する。
- ノード編集、エッジ編集、PNG/PDFエクスポートを追加する。

23. まとめ

このシステムは、Wikipediaという人間が作った知識リンク、TF-IDFという古典的な情報検索、Google Trendsという社会的な検索信号、NetworkXによるグラフ理論、Pydanticによるデータ検証、Ollama/Gemmaによる教育設計を組み合わせています。それぞれ単体では完璧ではありませんが、役割を分けて重ねることで、学習トピックから実用的なナレッジグラフを作る土台になっています。

> 一番大切な設計思想は、候補収集、関連度判定、学習順序化、保存、可視化を分離することです。この分離があるため、将来TF-IDFを埋め込み検索に替えたり、Gemmaを別モデルに替えたり、Web UIをReactに替えたりしても、全体を壊さずに進化させられます。