類似プログラムの提示ツール Selene

Selene: a Tool for Retrieving Relevant Programs

渡邉卓也 † 増原英彦 †
WATANABE Takuya Hidehiko MASUHARA
sodium@graco.c.u-tokyo.ac.jp masuhara@acm.org

† 東京大学大学院総合文化研究科
Graduate School of Arts and Sciences, The University of Tokyo

本研究では、統合開発環境にて編集中のプログラムに関係の深いコードを、開発者が参考に、あるいはプログラムの一部として利用出来るよう、貯蔵庫から自動的に検索し、提示するツール Selene を作成した。プログラムに特有の構文・意味情報を用いるツールとは異なり、Selene では連想計算技術を用いてコード全体の字面上の一致度を計算することにより、大規模な貯蔵庫から類似プログラムを高速に検索し提示することが可能となった。

1 はじめに

プログラムの作成中に類似したプログラムを参照することはその作成効率の向上に有効であり、多くのプログラマが日常的に行っている。参照の目的は多岐に渡り、既知の関数の細かい使用法を確認したい場合から、ある目的に利用可能な関数を知りたい場合や、対象分野のプログラムの一般的な作法を知りたい場合、あるいは既存のプログラムの一部分をそのまま複製して作成中のプログラムに組み込めそうな場合などが挙げられる。しかしながら、参照が必要な時に常に目的のプログラムが手元にあるとは限らず、むしろ汎用の検索ツールを用いる等して労力を掛けて検索しなければならない場合が多い。そこで、類似プログラムを検索し、適切な形でプログラマに提示するツールがあればプログラマにとって大きな助けになるものと考えられる。

実際類似プログラムの検索を助けるツールはいくつも提案されており、大きく二種類に分けることが出来る。

一つはプログラムの構文・意味上の手掛かりを用いて検索するツール群であり、多くは統合開発環境上で動作する。これらには、同じシグネチャを持つメソッドを検索するもの [15] や、指定された型のオブジェクトを生成する方法[*1]を提示するもの [9, 11]、あるいは指定されたクラスに似たクラスをフィールドや親クラスの型から検索[*2]して提示するもの [5] 等が含まれる。本論文ではこうしたツールをプログラム解析型ツールと呼ぶ。

もう一つは（一般には）少数の検索語（重要語、キーワード）を用いてそれを含むプログラムを探し出すツール群である。この中には単体で実行可能なツールの形で提供されているもの [2] もあれば、ウェブ上でのサービスとして提供されるもの [3, 8] もある。これらは元々自然言語を対象として開発されてきた検索技術を、対象領域をプログラムに限定して適用し利便性の向上を狙ったもので、基本的には検索語と検索対象との字面上の一致によって検索を行う。検索の際には、利用者は効果的と思われる検索語を自ら取捨選択して指定する。本論文ではこの型のツールを検索語一致型ツールと呼ぶ。

これらのツールには一長一短ある。プログラム解析型ツールについてはあまり労力を掛けずに比較的精度[*3]の高い検索結果を得ることが期待出来るが、プログラムの構文・意味についての解析に時間が掛かる為に大規模な貯蔵庫からの検索には適さない。また、特定の言語や、ある型のオブジェクトの生成法を知るといった限られた目的にのみ適用可能となり

[*1] 例えば、Java 言語において `FileInputStream` 型のオブジェクトから `BufferedReader` 型のオブジェクトを生成する等。

[*2] 同じ型のフィールドや共通の親クラスを持つクラスを類似していると判定する。

[*3] 検索結果中の、有用なものの割合。

がちである。具体的には現在では Java 言語を対象とし、貯蔵庫の大きさとしては数千クラスを想定、そこからある型のオブジェクトの生成方法を検索するといったツール [11] が典型的である。一方、検索語一致型ツールは高速な為大規模な貯蔵庫からの検索には適しているが、目的の類似プログラムを発見出来るような検索語の選定が難しく、精度が低い。加えて検索語の選定が利用者に任されており、もし一度の検索で有用なプログラムが発見出来なかった場合には、検索語を選び直して再び検索を行わせる必要があるので、手間が掛かる。

Selene はこれら二種類のツール群の中間点を狙いとしている。すなわち、プログラムの構文・意味情報は用いない代わりに、プログラム全体を検索語として用いてその字面上の情報を満遍なく利用する。これにより、大規模な貯蔵庫から高速に、特定の言語に依存せず、かつ利用者による明示的な検索語の指定無しにある程度の精度を保って検索することを可能とするものである。

2 Selene の特徴

以下 Selene の機能を具体例を用いて説明する。Selene は統合開発環境 Eclipse [1] のプラグインとして動作する。図 1 において、プログラマは Java 言語で Swing ライブラリを用いて GUI アプリケーションソフトウェアを開発中であり、プログラムは途中まで書かかれた状態となっている。現在プログラマはコンストラクタを定義中で、既に使用する予定の `JPanel` オブジェクトのインスタンス化部分を書き終えており、次にこれらを組み合わせ、また内部の配置を指定するつもりである。ここで `JPanel` クラスの利用方法をよく憶えていない場合、API 定義文書を調査することも可能であるが、このようによく使われる部品は自分で以前に同様の場面で使用したことがある可能性があり、また既に誰かが同じ様に使用している可能性も高いので、そうしたプログラムを参考に出来ると便利である。

Selene はエディタでの編集操作が行われた場合に自動的にその編集中のプログラムに類似したプログラムを検索（**連想検索機能、自動表示機能**）し提示するので、プログラマはこうした場合にすぐに参考になりそうなプログラムを参照することが出来る。図 1 下側には類似度の順に貯蔵庫中の類似プログラムのファイル名が表示されており、左側にはそれらのうち上位三つの、特に関連が深いと判定された一部分が表示（**関連部分表示機能**）されている。狙い通り、`JPanel` を利用している部分があり、左側上段からは `add()` メソッドによる `JPanel` オブジェクトの組み合わせ法、左側中段では `setLayout()` メソッドによる内部配置の指定法を読み取れる。プログラマはこれらを参考にして、あるいはそのまま複製して自分のプログラムを作成することが出来る。

もし参考になりそうなプログラムが提示されていなかった場合は、プログラム全体の情報を均一に利用するのではなく、カーソル行とその周辺行を重視して類似プログラムを検索（**行重み付き連想検索機能**）することも出来る。また、指定したプログラムの全体を取得してエディタの編集対象として開く（**プログラム取得機能**）ことも可能[*4]で、取得したプログラムから芋蔓式に検索を行うことが出来るので、比較的容易に目的のプログラムを見つけることが出来る。

以下では Selene のこのような動作を可能とする為の設計上の要点について述べる。このうち本研究の貢献は、

- 類似プログラムの検索
- 行重み付き連想検索
- 関連部分の表示

である。

2.1 連想検索

Selene では、類似したプログラムを検索する為に連想検索を用いている。連想検索とは、重要語集合ではなく、文書そのものを入力（検索語）として用いて類似文書を取得することをいう [13, p. 1]。連想検索の実現法は種々考えられるが、Selene では高野らによる汎用連想計算エンジン（Generic Engine for Transposable Association、以下 GETA）[4, 13] を利用し、連想計算による連想検索を行っている。連想計算とは、文書群同士や単語群同士、あるいは文書群と単語群間の類似性・関連性を算出することである [13, p. 1] が、ここでは特に、二文書間の類似度を

[*4] 関連部分が表示されているプログラムは既に全体が取得され、編集も可能な状態となっている。

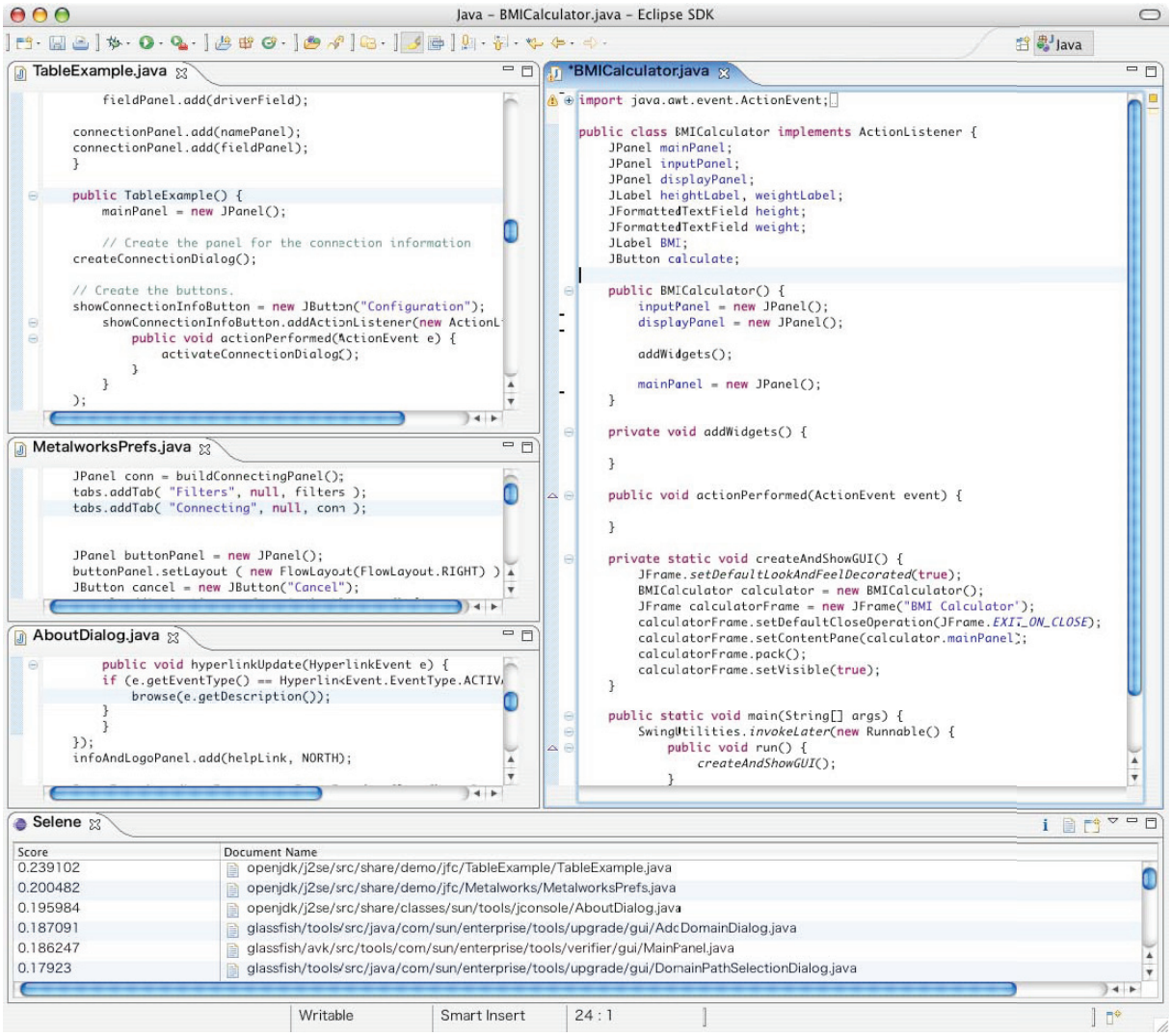

図 1　Selene の利用例

算出することをいう。文書同士の類似度は、文書を頻度情報付きの単語群とみて、その単語群同士の類似度を頻度情報を利用して計算することにより求める。この類似度算出の際の計算式を工夫することで、重要かつ特徴的な単語の頻度をより重く見て検索することが出来る。

Selene において連想計算を用いた理由は次のようなものである。プログラム全体を通して字面上類似したプログラムを検索しようとしたとき、単純にプログラムを文字列としてそのまま、検索語一致型ツールに入力として与えても満足な結果を得られないことは明白である。何故ならば、こうしたツールは完全一致を前提としているので、検索結果としては入力として与えたプログラムと同じ単語を全て含む極少数のプログラムしか得られないことが予想されるからである。一方、連想計算により類似度を求めれば、単語集合としては完全には一致しないプログラムも検索することが可能となるので、連想検索には都合がよい。また、検索結果の実用上の精度を向上させる為には、検索結果のプログラムそれぞれに対して適切に順位付けをすることが重要であるが、これは類似度を算出するという連想計算の性質により、その類似度順に並べることで実現することが出来る。

以降では文書同士の類似度の算出法につき説明する[*5]。算出法は Singhal らによる研究 [12] の方法をそのまま用いている。

連想計算における文書 j,k 同士の類似度 $\mathrm{sim}(j,k)$

[*5] 詳細に興味の無い場合は 2.2 節まで読み飛ばし可能。

は、単語の重みベクトル同士の内積であり、式 (1) によって求められる。

$$\mathrm{sim}(j,k) = \sum_i w_{ij} \times w_{ik} \tag{1}$$

ここで w_{ij} は単語 i の文書 j における重みである。なお、Selene における単語は、変数名、型名、メソッド名、コメント中の単語等、プログラム中に現れる文字列全てを指す。

単語の重みとしては TF-IDF（Term Frequency - Inverse Document Frequency) 重みがよく用いられ、これは典型的には式 (2) の形で表される [7, 14]。右辺の左側が TF 項であり、右側が IDF 項である。

$$w_{ij} = \underbrace{tf_{ij}}_{\mathrm{TF}} \times \underbrace{\log \frac{N}{df_i}}_{\mathrm{IDF}} \tag{2}$$

ここで tf_{ij} は単語 i の文書 j 中の頻度、df_i は単語 i を含む文書数、N は総文書数である。IDF 項を入れるのは、同時に多くの文書に共通して含まれる単語は高頻度語であっても一般的な単語であり、検索の際の絞り込みに有効な特徴的な語ではないと考えられる為である。例えば、`for` や `return` といった語があるプログラムに頻出していたとしても、そうした語は他の多くのプログラムにも含まれる一般的な語であり、検索の際に重視すべき語ではない。

実際の連想計算の際には、この TF-IDF 重みを様々な正規化項によって補正した重みが用いられ、Selene が現在採用している重み w'_{ij} は、Singhal ら [12] による以下のようなものである[*6]。

$$w'_{ij} = \frac{1 + \log tf_{ij}}{norm_j \times (1 + \log(average\ tf_j))} \times \log \frac{N}{df_i} \tag{3}$$

IDF 項は式 (2) と同一だが、TF 項が二つの正規化項によって正規化されており、一つは文書 j 中の単語の平均頻度である $average\ tf_j$、もう一つは回転異なり語数正規化項の $norm_j$ である。

$average\ tf_j$ によって tf_{ij} を正規化することにより、文書自体が大きい場合の tf_{ij} の増加を補正することが出来る。大きな文書の場合、単語の頻度は、もし出現確率が同一とするならば小さい文書に比べて多い。これを補正しないと大きな文書の類似度が過度に大きく算出されてしまう為、各単語の頻度を文書に含まれる単語の平均頻度で割ることで正規化を行っている。

回転異なり語数正規化項 $norm_j$ は、異なり語数[*7]による正規化項の、文書の大きさに応じて発生する理想的な値からのずれを補正したもので、以下の式により定義される。

$$norm_j = (1.0 - slope) \times pivot + slope \times ut_j \tag{4}$$

$slope$ は ut_j により定義される正規化直線に対する回転の度合い、$pivot$ は回転の軸であり、ut_j は文書 j 中の異なり語数である。$slope$ には適当な定数、$pivot$ としては全文書中の異なり語数の平均を用いている。異なり語数で正規化するのは、大きな文書は多くの異なる単語を含む為、類似度の算出の際に多くの単語の重みの積和を取ることとなり、そのままでは大きな文書に対して過度に大きな類似度を与えてしまうからである。

2.2 行重み付き連想検索

Selene ではプログラム全体を均一に扱う検索だけでなく、カーソル行を中心にした周辺行の単語をより重視して検索を行うことも可能である[*8]。これは、検索の利用目的によっては、プログラム中の単語の情報を均一に利用するよりも特定の箇所をより重視して検索を行った方がより良い結果を期待出来る場合がある為である。例えば、ファイル入出力関係のクラスの利用法を知りたいという時に、プログラムの他の部分は主に GUI に関する操作を行っていて、そのままでは GUI 関連のプログラムばかり表示されてしまい、有用な結果を得られないといった場合である。

行 l の重み w_l は式 (5) によって算出され、カーソル行から離れる程単調に減少する。

$$w_l = \frac{1}{1 + d_l \times (1 - c_{att})} \qquad (c_{att} \leq 1) \tag{5}$$

ここで d_l はカーソル行からの距離、c_{att} は減衰係数である。式 (5) は 3.2 節にて述べる実装上の理由により、$0 < w_l \leq 1$ となるように定義されている。c_{att} が

[*6] この重み付けは自然言語処理分野において有効であることが示されている [12, 13]。

[*7] 異なり語数とは、ある文書中の異なる単語の数である。

[*8] 特定の範囲を指定して検索することも可能。

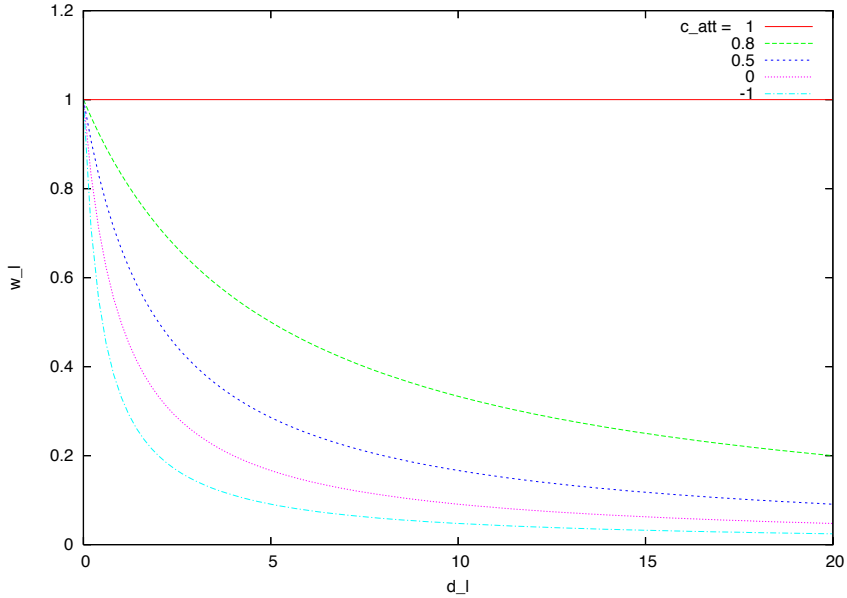

図 2　行重みの減少の様子

1 のときは全ての行を均等に扱う通常の連想検索と同じとなり、c_{att} を小さくする程重み曲線の傾斜が急になる。重みの減少の様子を図示したのが図 2 である。この c_{att} は利用者が随時変更可能であり、検索の目的に応じて自由に調整することが出来る。

2.3　自動表示

Selene では、検索語の明示的な入力無しに検索を自動的に行い結果を表示することで、プログラマが有用な結果が得られると予期していない場面についてもプログラマの助けとなることを狙いとした。Ye ら [15] によって、自動的な検索結果の表示によってプログラマを助けることが有用であることが示されている。

この自動的な検索・提示を行うのに適した環境は統合開発環境であり、Selene は統合開発環境 Eclipse のエディタで編集されているプログラムに類似のプログラムを貯蔵庫から自動的に検索し提示する。編集されているプログラムにある程度の変更が加えられた時に自動的に再検索を行う[*9]ので、プログラマの関心の変化に応じた検索結果を得ることが出来る。（勿論任意の時点での検索も可能である。）

[*9] 行わないように設定することも出来る。

2.4　関連部分の表示

Selene は検索結果中の類似度上位のプログラムにつき、編集中のプログラムに最も関連が深いと判定された一部分を表示する。これは次のような手順による。まず、連想検索により編集中のプログラム p_1 に類似した貯蔵庫中のプログラムの集合 $\mathbf{P}_1$ が得られたとする。この $\mathbf{P}_1$ 中の、類似度上位のあるプログラムを p_2 とする。p_2 につき、全文を貯蔵庫から取得して予め定められた大きさに区切る。区切られた各部分を $p_{2i}\quad (i = 1, \ldots, n)$ とする。それぞれの部分 p_{2i} を検索語として、最初の連想検索と同じ貯蔵庫に対し連想検索を行う。すると、部分 $p_{2i}\quad (i = 1, \ldots, n)$ それぞれにつき類似したプログラムの集合 $\mathbf{P}_{2i}\quad (i = 1, \ldots, n)$ が得られるので、各々 $\mathbf{P}_1$ との積をとる。これによって得られた積集合の大きさを比較し、最も大きな積集合を持つ部分を、最も p_1 と関連が深い部分であると判定する。

3　実装

以下 Selene の実装における全体の構成と、実装上の特徴を述べる。

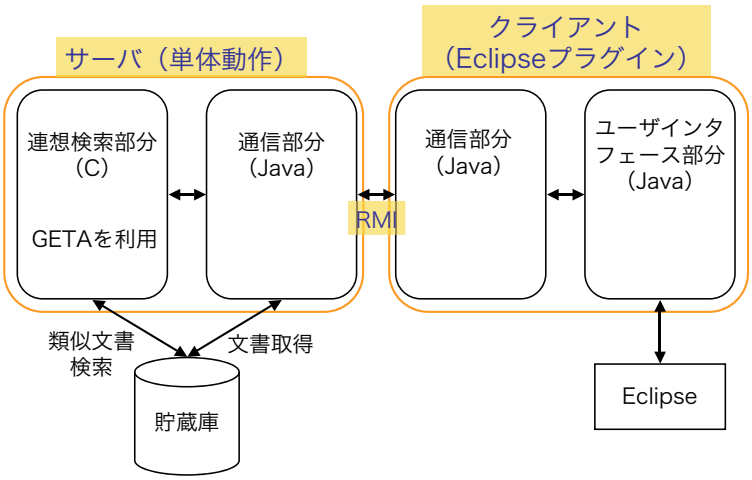

図 3　Selene の全体の構成

3.1　全体の構成

Selene の全体の構成は図 3 のようになっている。Selene はサーバとクライアントに分かれて動作することによって貯蔵庫側と開発環境側を分離する構成となっている。これにより貯蔵庫を複数の開発者で共有して開発を行うといったことが容易に出来る。

サーバは更に実際に連想検索を行う部分（連想検索部分）とクライアントとの通信を受け持つ部分（通信部分）に分かれ、またクライアントはサーバとの通信を行う部分（通信部分）とユーザインタフェースを担当する部分（ユーザインタフェース部分）に分かれる。使用したプログラミング言語により分類すると、連想検索部分のみが C 言語で、残りは Java 言語である。動作環境としては、サーバは単体で動作、クライアントは Eclipse のプラグインとして常に Eclipse 上で動作する。プログラムの規模は、プログラム全体の行数がおおよそ 1700 行で、Java 言語で書かれた部分のクラス数（無名クラスを除く）が 22 クラスである。

一回の検索は次のようにして実行される。まず、自動もしくは利用者の明示的な指定によって、Eclipse のエディタより、編集中のプログラムが検索語としてユーザインタフェース部分により取得される。このプログラムは現在のカーソル位置等の情報を付加された後、クライアントおよびサーバの通信部分を経由して、連想検索部分に渡される。連想検索部分では渡されたプログラムは単語に分解され、各単語の頻度が集計される。この頻度情報付きの単語群と貯蔵庫中のプログラムとの連想計算が行われ、貯蔵庫中のプログラムのファイル名が類似度順に並び替えられて連想検索部分から出力される。ファイル名一覧は再び通信部分を経由してユーザインタフェース部分に渡され、画面に表示される。なお、これらの処理は全て Eclipse のユーザインタフェース周りのスレッドとは非同期に行われるので、検索の実行中も利用者はプログラムの編集等を行うことが出来る。

利用者の指定したプログラムを取得する機能の場合は連想検索部分が実行されないが、ユーザインタフェース部分に始まる処理の流れは基本的に同一である。

各部分間の通信は、連想検索部分とサーバの通信部分間は標準入出力のパイプ結合により、サーバとクライアントの通信部分間は RMI により、クライアントの通信部分とユーザインタフェース部分間は通常のメソッド呼び出しによって行われる。

3.2　連想検索部分

連想検索部分では既に述べたとおり高野らによって開発された GETA [4, 13] を利用している。連想計算を行う為にはプログラム中に含まれる単語の頻度を集計しなければならないので、渡されたプログラムはまず単語毎に分解される。この過程で記号類の除去や、小文字への揃えが行われる。その結果を元に頻度集計が行われ、その後 GETA のライブラリによる連想計算が行われる。それにより、渡されたプ

ログラムと貯蔵庫中のプログラムの類似度が算出されるので、貯蔵庫中のプログラムのファイル名を類似度順に並び替えて出力する[*10]。

行重み付き連想検索は、プログラム中の単語頻度の集計時に、単語の出現毎にその単語の頻度を 1 増やすのではなく、代わりにその単語の位置する行の重み w_l を足していくことによって実現している。従って $w_l \geq 0$ でなければならず、さらに、$w_l > 0$ とすれば、カーソル行から遠い位置にある低頻度語の影響を後で調整可能となる。また式 (5) のように定義し、最大で 1 としておけば、$c_{att} = 1$ の場合に通常の連想検索と全く同じ結果を得ることが出来るので処理部分を共通化可能である。

結果、$0 < w_l \leq 1$ なので、行重み付きで集計した頻度は、行重み無しでの集計の場合に比べ、小さくなる。これは、より小さなプログラムから検索した場合に相当するものの、プログラムの大きさに関しては正規化項によって検索結果の変動を補正しているので、それ程影響は無いものと思われる。

この手順によって得られた単語の頻度は実数になるが、GETA は整数として単語の頻度を扱うので、何らかの閾値によって丸めなければならない。この際の丸めの閾値は利用者によって自由に設定可能としており、低頻度語をどの程度検索の際に考慮するかの微調整に利用可能である。

3.3 関連部分の表示

関連部分表示の為の処理は、最初の連想検索の結果が返って来た後に開始される。処理の流れは 2.4 のとおりであるが、類似度上位のプログラムそれぞれ（図 1 では上位三つ）についての関連部分の表示処理は並行して行われる。従って結果が表示される順番は類似度の順番という訳ではなく、主として連想計算に掛かる時間、すなわちプログラムの大きさに依存する。

関連部分の場所を特定した後は、Eclipse のエディタを用いてプログラムを開き、その場所にカーソルを移動させることで関連部分の表示を行っている。よって表示されていない部分を見たり、何らかの編集操作を行ったりということは自由に出来る。

[*10] 全て出力する訳ではなく、予め指定された数で打ち切る。

表 1 貯蔵庫の規模

ファイル数	行数	単語数	バイト数
1000	161,901	525,916	5,117,983
5000	986,072	3,558,988	33,242,573
10000	2,070,338	7,243,279	70,509,314
20000	4,486,042	16,548,465	153,693,878
30000	7,337,992	27,723,215	262,658,979
(57127)	13,452,995	51,911,209	485,612,469

4 評価

Selene がどの程度大規模な貯蔵庫に対して適用可能なのかを調査する為、実験を行った。実験は貯蔵庫の大きさを 1000 ファイルから 30000 ファイルまで 5 段階に変化させて行い、それぞれの大きさの貯蔵庫につき、無作為に抽出したプログラムを検索語とした連想検索を実行し、実行時間を計測した。

各貯蔵庫は、約 57000 ファイルからなる、Java 言語で書かれた原始ファイルの貯蔵庫から無作為にファイルを抽出することで構築した。原始ファイルは Eclipse や OpenJDK [10] 等のフリーソフトウェアのものを利用した。各貯蔵庫の規模は、表 1 に示すようなものである。ファイル数 10000 の貯蔵庫でプログラムの行数は二百万行に達しており、これは現時点での大規模プロジェクトの原始ファイルと同等の量である。

50 回実行した平均の実行時間は、図 4 に示すとおりとなった。サーバは Pentium Xeon 3 GHz×2、メモリ 6 GB で、OS は Linux である。また、クライアントは PowerPC G5 2 GHz×2、メモリ 3 GB で、OS は Mac OS X である。サーバとクライアントは 1000BASE-T の LAN で結ばれている。

最初の連想検索によって類似プログラムのファイル名一覧を表示するまでの時間は、ファイル数 30000 においても 1 秒以内に納まっており、十分な速度といえる。ファイル数に応じた実行時間は、概ね線形に増加しているので、類似プログラムのファイル名を表示するだけであればこれ以上の規模の貯蔵庫を利用することも可能である。関連部分表示について

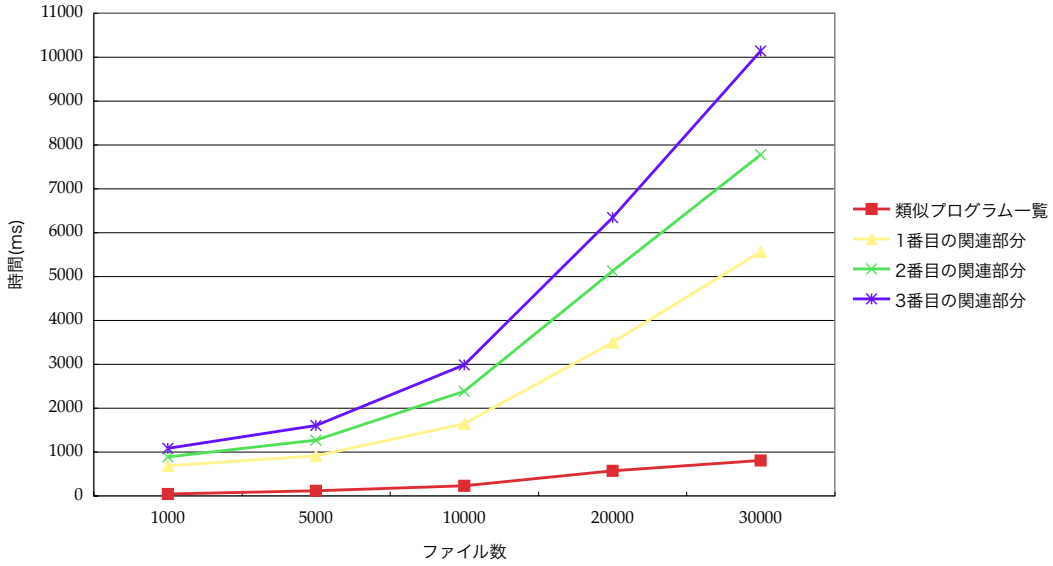

図 4　貯蔵庫の大きさと検索に掛かる時間の関係

は、ファイル数 30000 では 1 番目の関連部分が表示されるまで平均 5.6 秒、3 番目の関連部分の表示までには平均 10.1 秒掛かっているので、これより大規模な貯蔵庫の利用に際しては十分な速度が出ない可能性がある[*11]。ファイル数 10000 の場合について見ると、3 番目の関連部分が表示されるまでに平均 3.0 秒で済んでおり、従って、大規模プロジェクトの原始ファイルから検索するような用途においては十分な実行速度であると言うことが出来る。

5　関連研究

Ye らによる CodeBroker [15] は、プログラムの作成中にメソッドのシグネチャを入力した時点で、それに類似するメソッドを自動的に検索し、メソッド名とシグネチャの一覧を表示する。メソッド同士の類似度は、シグネチャと、それより前に書かれたコメント文の類似度により算出される。

Holmes らによる Strathcona [5] は、利用者の指定したクラスやメソッド等に関連したコードを貯蔵庫から検索してプログラマに提示するものである。Selene と同様に Strathcona もサーバとクライアントに分かれ、貯蔵庫からの検索はサーバが行う。クライアントはコードの置かれている構造的な文脈、つまり親クラスやフィールドの型といった情報を抽出し、サーバに送る。サーバはその構造的文脈を様々なヒューリスティクスを用いて貯蔵庫中のコードと照合し、合致したものをクライアントに送り返す。

Mandelin らによる Prospector [9] は、カーソルの置かれている場所で必要な型への、様々な型からの変換過程を提示する。この変換過程はジャングロイド（jungloid）と呼ばれ、型から型への、メソッドを辺のラベルとした有向グラフの形で表現される。ジャングロイドは事前に既存のコードから抽出され保存されているものを利用する。

Sahavechaphan らによる XSnippet [11] は Prospector と同じ用途のツールだが、Strathcona のように構造的文脈を利用して検索精度の向上を図っている。

検索語一致型ツールとしては、gonzui [2]、Koders [8]、Google Code Search [3] 等がある。

他、Kamiya らによる CCFinder [6] は、主にコードの複製によって生じる重複コード（コードクローン）を探すことを目的としている。Selene がプログラム中の単語の出現順序を考慮しないのに対し、CCFinder は同一順序で並ぶ単語列を発見する為のツールである。

[*11] 本実験では、関連部分を検索する為にプログラムを 20 行ずつに区切って連想検索を行っている。この区切る単位を大きくすると、関連部分の表示はより高速になる。

6 まとめと今後の課題

本論文では、類似プログラムの提示ツール Selene を提案し、そのスケーラビリティにつき評価を行った。Selene はプログラムの構文・意味情報を用いず、代わりにプログラム全体を検索語として用いる連想検索によって字面上の情報を満遍なく利用する。これにより、大規模な貯蔵庫から高速に、特定の言語に依存せず、かつ利用者による明示的な検索語の指定無しにある程度の精度を保って類似プログラムを検索出来る。連想検索には連想計算を利用し、検索語と貯蔵庫中のプログラムとの類似度を算出した。また、行重み付きの連想検索や関連部分の表示を実現した。実験によるスケーラビリティの評価では、大規模な貯蔵庫に対して十分な速度での検索を行えることを示した。

今後の課題としては、関連部分表示の速度を向上させること、類似度算出の計算式をプログラムに適したものに改良すること、検索精度の評価を行うことが挙げられる。

謝辞

関連部分表示の実現法につき貴重な提案を下さった、国立情報学研究所の高野明彦氏に深く感謝致します。

参考文献

[1] Eclipse, http://www.eclipse.org/

[2] gonzui, http://gonzui.sourceforge.net/

[3] Google Code Search, http://www.google.com/codesearch

[4] 汎用連想計算エンジン（GETA）, http://geta.ex.nii.ac.jp/

[5] Reid Holmes, Gail C. Murphy, 2005, Using structural context to recommend source code examples, ICSE '05: *Proceedings of the 27th International Conference on Software Engineering*, pp. 117–125

[6] Toshihiro Kamiya, Shinji Kusumoto, Katsuro Inoue, 2002, CCFinder: a multilinguistic token-based code clone detection system for large scale source code, *IEEE Trans. Softw. Eng.*, Vol. 28, No. 7, pp. 654–670

[7] 金明哲, 村上征勝, 永田昌明, 大津起夫, 山西健司, 2003, 『言語と心理の統計』（統計科学のフロンティア 10）, 岩波書店, pp. 88–89

[8] Koders, http://www.koders.com/

[9] David Mandelin, Lin Xu, Rastislav Bodík, Doug Kimelman, 2005, Jungloid mining: helping to navigate the API jungle, PLDI '05: *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 48–61

[10] OpenJDK, http://openjdk.java.net/

[11] Naiyana Sahavechaphan, Kajal Claypool, 2006, XSnippet: mining for sample code, OOPSLA '06: *Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications*, pp. 413–430

[12] Amit Singhal, Chris Buckley, Mandar Mitra, 1996, Pivoted document length normalization, SIGIR '96: *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 21–29

[13] 高野明彦, 西岡真吾, 今一修, 岩山真, 丹羽芳樹, 久光徹, 藤尾正和, 徳永健伸, 奥村学, 望月源, 野本忠司, 2002, 「汎用連想計算エンジンの開発と大規模文書分析への応用」, http://geta.ex.nii.ac.jp/pdf/itx2002.pdf

[14] 徳永健伸, 1999, 『情報検索と言語処理』（言語と計算 5）, 東京大学出版会, pp. 27–32

[15] Yunwen Ye, Gerhard Fischer, 2002, Supporting reuse by delivering task-relevant and personalized information, ICSE '02: *Proceedings of the 24th International Conference on Software Engineering*, pp. 513–523