コメント中の頻出文字列を用いたソフトウェアライセンスの特定支援

Semi-automatic identification of software license using frequent string in code comment

真鍋 雄貴*　市井 誠†　早瀬 康裕‡　松下 誠§　井上 克郎¶

あらまし　ソースファイル検索システムを用いることで，開発者は再利用に適したソースファイルを容易に取得できる．しかし，取得したソースファイルの使用条件 (ライセンス) を特定する作業は開発者が手作業で行なわねばならず，再利用にかかる労力を大きくしていた．この労力を削減するには，予め検索システムに登録されるソースファイルのライセンスを特定しておき，検索結果でソースファイルのライセンスを表示するのが望ましい．本研究では，ソースファイルのライセンス特定を，検索システムの管理者が効率的に行うためのシステムを提案する．提案するシステムは，複数のソースファイル中のコメントに頻出する文字列を抽出する．ライセンスはソースファイル中のコメントとして記述されることが多いため，管理者はこの文字列を読むことで，一度に多くの部品のライセンスを特定することが出来る．提案するシステムを評価するために複数のオープンソースソフトウェアを対象とした実験を行ない，ライセンス特定のカバー率や，特定に要した時間を計測した結果，本手法の有効性が示された．

1　はじめに

ソフトウェア開発コストの削減に有効な方法の１つとして，ソフトウェアの再利用が挙げられる [10]．再利用の対象となるソフトウェア資産として，本研究ではソースファイルに注目する．開発者が再利用を行うためには，まず必要なソースファイルを膨大な量のソフトウェアから選択して取得する必要がある．開発者が利用可能なソフトウェアとしては，過去に開発者が自ら開発したソフトウェアに加え，SourceForge.net [8] 等で開発されるオープンソースソフトウェアがある．

このような膨大な量のソフトウェアからのソースファイルの取得を支援するシステムとして，ソースファイル検索システムがある [3] [5] [6] [14]．ソースファイル検索システムにはオープンソースソフトウェアなどから得られたソースファイルが蓄積されており，開発者は検索クエリを入力することにより必要なソースファイルを取得することができる．

しかし，取得したソースファイルを利用するためには使用条件（ライセンス）を特定せねばならず，これにかかる労力が再利用の妨げとなっている．開発者がライセンスを特定するためには，ソースファイルやソースファイルに関連する文書を読む必要がある．また，近年のオープンソースソフトウェア開発の普及にともない，多様なライセンスが出現しており [7]，開発者がライセンスを特定する作業を困難にしている．

このような開発者の労力は，ソースファイルを取得するのと同時にライセンスを知ることが出来れば不要である．そのためには，ソースファイル検索システムに蓄積されたソースファイルのライセンスを予め特定しておく必要がある．しかし，ソー

*Yuki Manabe, 大阪大学 大学院情報科学研究科

†Makoto Ichii, 大阪大学 大学院情報科学研究科

‡Yasuhiro Hayase, 大阪大学 大学院情報科学研究科

§Makoto Matsushita, 大阪大学 大学院情報科学研究科

¶Katsuro Inoue, 大阪大学 大学院情報科学研究科

スファイルの数は膨大であり，またライセンスも多様であるため，ライセンスの特定を全て人手で行うのは現実的では無い．

そこで，本稿では，大規模なソースファイル集合に対して適用可能なライセンス特定支援手法を提案する．我々は，ソースファイルに対するライセンスの指定がそのソースファイルのコメントに記述されることが多く，また，その記述には共通する文字列表現が用いられる場合が多いことに着目した．そこで，提案手法は，コメント中での出現頻度の高い文字列（共通文字列）を抽出し，抽出された文字列に対してライセンスを結び付けることにより，多数のソースファイルのライセンスを効率的に特定する．

提案手法の有効性を検証するために適用実験を行い，ライセンス特定の効率，特定に要する時間，利用者の労力の 3 つの観点で評価した．まず，利用者が読んだ共通文字列の数と，ライセンスが特定されたソースファイルの数を計測した結果，少数の共通文字列を読むだけで大部分のソースファイルのライセンスを特定できることを確認した．続いて，手順が終了するまでに要する時間を評価した結果，提案手法が実用的な時間で終了することが示された．最後に，手順が終了するまでに利用者が読む必要のある共通文字列数より，利用者の労力を評価した結果，共通文字列のフィルタリングなどによる改善の余地があると分かった．

以降，2 節では提案するライセンス特定手法の説明を行なう．3 節では提案手法の評価のため，実際のオープンソースソフトウェアを実験対象とした適用実験の内容を説明する．4 節では適用実験の結果に基づき提案手法について考察する．また，提案手法の制限についても述べる．5 節では関連研究の紹介を行ない，6 節ではまとめと今後の課題を述べる．

2 提案手法

本節では，提案する複数のソースファイルのコメント間に共通して出現する文字列を用いた，ライセンス特定の支援手法を説明する．ライセンスとは，著作権者がソースファイルの利用希望者に対して定めた，そのソースファイルを利用するための条件のことを表す．

2.1 ライセンスの指定方法

あるソースファイルに適用されるライセンスの指定は，主に，次に示す 2 通りの方法により行われる．ひとつはソースファイル中にコメントとして記述する方法であり，もうひとつはドキュメントファイルに記述する方法である．以下，それぞれについて説明する．

まず，ソースファイル中にコメントとして記述する方法では，コメントにライセンスの文面，もしくはそれを記述したファイルの場所が記述される．このようなライセンスを指定するコメントは，１つのソフトウェアの配布パッケージに含まれる同一ライセンスのソースファイルにおいて，共通している場合が多い．例えば，オープンソースソフトウェアのプロジェクトを多く抱える Apache Software Foundation [1] では，ソースファイルの先頭部分に共通したライセンス記述を含めることを規約としている [2].

一方のドキュメントファイルに記述する方法では，ソフトウェアの配布パッケージに含まれるソースファイルのライセンスが，パッケージ中の COPYING という名前のファイルなどに記述され，ソースファイルにはライセンスに関する記述は行われない．

一般に，ドキュメントファイルに記述する方法では，個々のソースファイルを取り出した場合にライセンスの記述が失われてしまうため，ソースファイル中にコメントとして記述する方法が用いられることが多い．

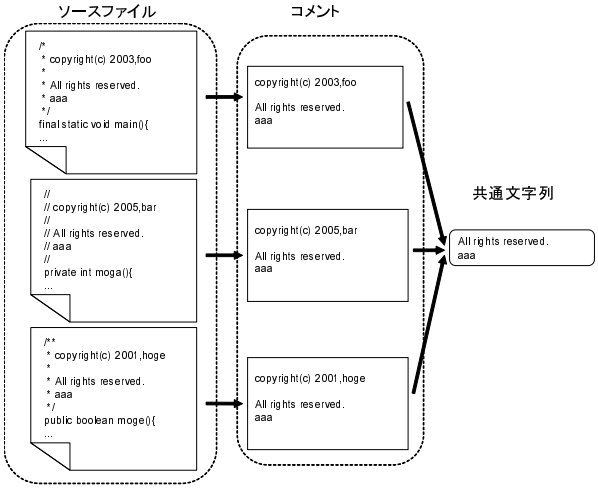

図 1　共通文字列の抽出

2.2　共通文字列を用いたライセンス特定

本研究では，ソースファイル中にコメントとして記述する方法において，共通した記述が用いられることが多い点に着目する．我々はこの着目点に基づき，複数のソースファイルのコメントに共通して含まれる文字列を抽出することにより，ライセンスを指定する記述が抽出されると考えた．ここで，ライセンスを指定する記述とは，ソースファイルの利用条件を示す文字列，もしくは，それを記述しているファイルがある場所を示す文字列とする．本稿では，複数のソースファイルのコメントに共通して含まれる文字列を共通文字列と呼ぶ．ソースファイルからどのような共通文字列が抽出されるかを表す例を図 1 に示す．図 1 では，3 つのファイルからそれぞれコメントの部分が抽出され，そして，3 つのコメントに共通して現れている「All rights reserved. aaa」が共通文字列として抽出されている．この例の場合，利用者が共通文字列「All rights reserved. aaa」を読み，この共通文字列が示すライセンスを特定することが出来れば，3 つのソースファイルに対して，それらソースファイルのライセンスを特定することができる．ここで，共通文字列はライセンスを特定する記述を含まない場合がある．抽出された共通文字列は以下の 3 種類に分類される．

1. ライセンスを指定する記述を含み，その記述が示すライセンスを特定することが可能である
2. ライセンスを指定する記述を含み，その記述が示すライセンスを特定することが不可能である
3. ライセンスを指定する記述を含まない．

共通文字列の分類は，共通文字列に含まれる，ライセンス名やライセンスを指定する記述を含むファイルへの参照，URL，列挙されている条項に基づいて判定する．2. は，複数のライセンスを指定する記述の部分文字列となっている共通文字列が該当する．

このような共通文字列は大量に抽出されるため，利用者がどの共通文字列を読むべきかを決める基準が必要となる．本研究では，この基準として未特定ファイル数という尺度を提案する．未特定ファイル数とは，ある共通文字列を含むソースファイルのうち，ライセンスが特定されていないソースファイルの数である．R を，ソースファイル f と f のライセンス l との関係 (f, l) を要素とする集合であるとすると，共通文字列 p の未特定ファイル数は以下の unknownLicenseFiles 関数により求めら

1. ソースファイル集合 F から連結コメント集合 C を抽出
2. C から共通文字列集合 P_0 を抽出
3. $P = P_0$
4. **while**$(P \neq \phi)$
5. 集合 $\{p' | p' \in P \wedge \text{unknownLicenseFiles}(p') = \max_{p \in P} \text{unknownLicenseFiles}(p)\}$ の要素を 1 つ選び，$p_{\max}$ とする
6. 利用者は $p_{\max}$ を読む
7. 利用者が $p_{\max}$ が特定可能なライセンスを指定する記述を含むか真偽値を入力
8. **if**(7. で入力された値が真である)
9. 利用者はライセンス l を入力
10. $L \leftarrow L \cup \{l\}$
11. $R \leftarrow R \cup \{(f', l) | f' \in \text{relatedFile}(p_{max})\}$
12. **end if**
13. $P \leftarrow P \backslash (\{p | \text{unknwonLicenseFiles}(p) = 0\} \cup \{p_{\max}\})$
14. **end while**

図 2　提案手法におけるソースファイルのライセンスを特定する手続き

れる．

$$\text{unknownLicenseFiles}(p) = |\{f | \forall l, (f, l) \notin R \wedge f \in \text{relatedFile}(p)\}|$$

relatedFile(p) は p を含むソースファイル集合を返す関数である．例として，図 3 のような対応関係があるとする．図 3 では，ソースファイルとコメント，共通文字列の対応関係の一例を示している．この図において，矢印は対応関係を表し，A→B は A は B に含まれる文字列であることを示す．この図 3 の例では，$\text{relatedFile}(p_2) = \{f_1, f_2, f_3, f_4\}$ である．

また，ライセンスが特定されたソースファイルとは，ソースファイルが含む共通文字列のうち，利用者により共通文字列が示すライセンスが特定された共通文字列が 1 つ以上存在するソースファイルとする．未特定ファイル数は，利用者がその共通文字列が示すライセンスを特定することができれば，新たにライセンスが特定されるソースファイルの数を意味する．

提案手法では，利用者が，共通文字列が示すライセンスを特定する．理由は，結果の信頼性を重視し，ソースファイルとライセンスの関係に誤りを含まないようにするためである．パターンマッチなどの方法により，その記述がどのライセンスを指定するかの解釈を自動的に行うことも考えられるが，完全に誤りを防ぐことは困難である．得られたライセンスの特定結果に誤りを含む場合は，部品を再利用するときに改めてライセンスが正しいか調査する必要があり，部品を再利用するための開発者の労力を軽減するという本研究の目的を達成できない．

2.3　ライセンス特定の手順

提案手法は，ソースファイル集合 F を入力とし，ソースファイル f とライセンス集合 L に属するライセンス l の関係 (f,l) の集合 R を得る対話的な手続きである．なお，L は手続き開始時には空集合である．

具体的なライセンス特定の手順を図 2 に示す．以下，図 2 の手順について詳細に説明する．

2.3.1　コメントの抽出

ソースファイル集合 F を入力として，連結コメントの集合 C を抽出する (図 2, 行 1)．連結コメントとは，ソースファイルに含まれる全てのコメントを 1 つの文字列に連結した文字列である．この処理では，ひとつのソースファイル中の全てのコメントはひとつの連結コメントとして抽出される．そのため，F に属する各ソースファイルからそれぞれ 1 つの C に属する連結コメントが抽出され，F の各要素と C

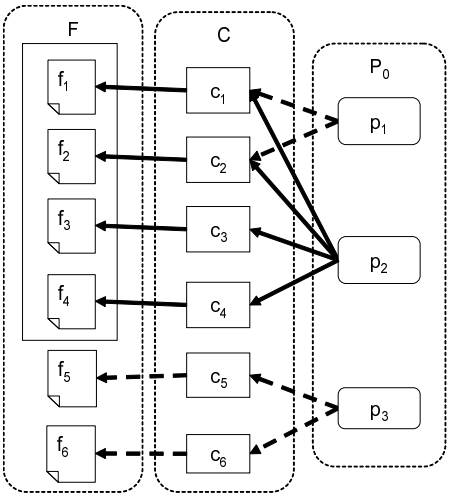

図 3　共通文字列とソースファイルの対応

の各要素は 1 対 1 に対応する．

2.3.2　共通文字列の抽出

連結コメント集合 C を入力として共通文字列集合 P を抽出する (図 2，行 2)．P に属する共通文字列は C に属する複数の連結コメントから抽出されるため，P の要素 1 つに対し．C の要素が複数対応する．一方，C に属する連結コメント 1 つに複数の共通文字列が含まれている場合があるため，C の要素 1 つに対し．P の要素が複数対応する．したがって，C の要素と P の要素には多対多の対応関係がある．

2.3.3　手作業によるライセンスの特定

利用者が共通文字列を読み，共通文字列にライセンスを指定する記述が含まれると利用者が判断する場合，ライセンスを入力することにより，読んだ共通文字列を含むソースファイルのライセンスを特定する．まず，利用者は表示された共通文字列を読む (図 2，行 6)．次に，利用者は共通文字列がライセンスを指定する記述を含むかどうか入力する (図 2，行 7)．最後に，共通文字列がライセンスを指定する記述を含んでいると利用者が入力した場合 (図 2，行 8)，利用者は指定する記述がどのライセンスを示すか特定し，そのライセンスを入力する (図 2，行 9)．ライセンスが入力されると，手続きは利用者が入力したライセンスをライセンス集合 L に追加する (図 2，行 10)．そして，利用者が入力したライセンスと利用者が読んだ共通文字列に対応するソースファイル集合との関係が R に追加される (図 2，行 11) ことにより，読んだ共通文字列に対応するソースファイルのライセンスが特定される．

2.3.4　終了条件

手続きの終了条件は集合 P が空集合であることである (図 2，行 4)．1 つの共通文字列を利用者が読むたび，P から利用者が読んだ共通文字列と未特定ファイル数が 0 である共通文字列が除去される (図 2，行 13)．未特定ファイル数が 0 である共通文字列を除去する理由は，利用者が未特定ファイル数 0 の共通文字列がライセンスを特定可能な記述を含み，そのライセンスを特定したとしても，新しくライセンスが特定されるソースファイルの数は 0 であるためである．

3　適用実験

提案手法の有効性を評価するため，オープンソースソフトウェアに含まれるソースファイルの集合を用いた適用実験を行った．

3.1 提案手法の有効性評価方法

本実験では，ライセンス特定の効率，利用者が共通文字列を読む労力，ライセンスの特定にかかる時間の 3 つの観点から提案手法を評価する．

実験では複数のオープンソースソフトウェアに含まれるソースファイルの集合 4 組に対して，提案手法を適用した．それぞれの実験対象を構成しているソフトウェア，それらのソフトウェアが基づくライセンス，対象ソースファイル数を表 1 に記す．なお，実験では Java ソースファイルのみを対象とした．実験対象 1，実験対象 2，実験対象 4 は異なったソースファイル数で構成されており，ソースファイル数が増えた場合の利用者が共通文字列を読む労力や，ライセンスの特定にかかる時間の変化を評価することができる．また，実験対象 2 と実験対象 3 はソースファイル数に大きな差は無いが，互いにライセンスが重複しないソフトウェアで構成してあり，ライセンスの違いが提案手法に与える影響を確認することができる．

提案手法の効率を評価するため，ライセンス特定のカバー率を評価する．カバー率はライセンスを特定できたソースファイル集合を $F_{\mathrm{identified}}$，ライセンス特定の対象とするソースファイル集合を F としたとき，以下のように定義される．

$$\text{カバー率} = \frac{|F_{\mathrm{identified}}|}{|F|}$$

次に，ライセンス特定に要した時間を測定する．

最後に，利用者が何個の共通文字列を見る必要があるかを測定する．

実験は，提案手法の簡単な実装を行い，第一著者が表 2 で示す計算機環境を用いて行った．共通文字列の抽出には CCFinder [12] を用いた．

表 1　実験対象

ソフトウェア名	配布ライセンス	ファイル数
実験対象 1		
Freemaker	BSD License	318
galleon	GPL	236
	合計	554
実験対象 2		
JTrac	Apache License V2.0	156
JUNG	BSD License	647
jEdit	GPL	520
FindBugs	LGPL	818
	合計	2141
実験対象 3		
JMRI	Artistic License	985
Junit	Common Public License	173
RSSOwl	Eclipse Public License	413
mged	MIT License	467
XUI	MPL 1.1	310
	合計	2348
実験対象 4		
Artifactory	Apache License V2.0	158
Spring Framework	Apache Software License	3086
Jetty	Artistic License	461
HSQL Database Engine	BSD License	131
Robocode	Common Public License	172
Bioclipse	Eclipse Public License	1288
ZK	GPL	804
Hibernate	LGPL	1556
Jester	MIT License	55
JGAP	MPL 1.1, LGPL	108
	合計	7819

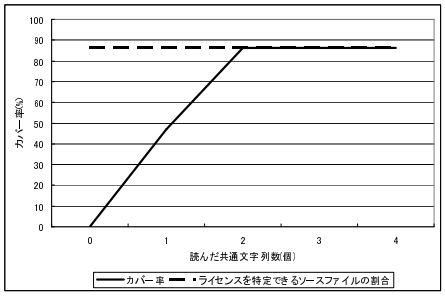

図 4　実験対象 1 を対象としたカバー率

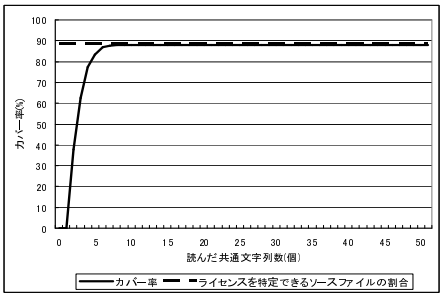

図 6　実験対象 2 を対象としたカバー率

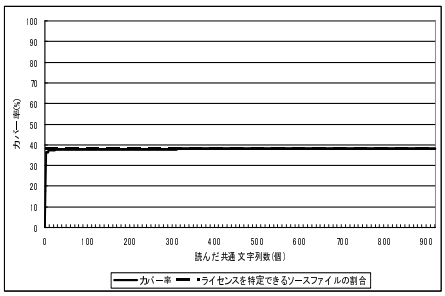

図 5　実験対象 3 を対象としたカバー率

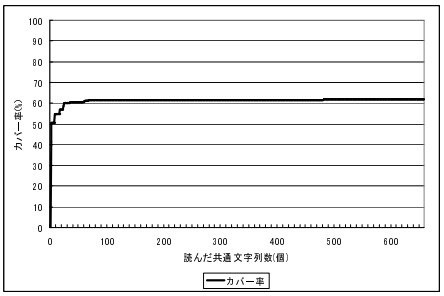

図 7　実験対象 4 を対象としたカバー率

3.2　測定結果

3.2.1　評価 1:カバー率の測定

実験対象 1〜4 に提案手法を適用したときの，カバー率の推移をそれぞれ図 4〜7 に示す．各グラフにおいて，実線はカバー率の推移，破線は，実験対象のソースファイルに占めるライセンスを特定できるファイルの割合を示している．ここで，ライセンスを特定できるファイルとは，ライセンスを特定可能な記述を含むファイルのことを指す．この割合は，時間の都合により実験対象 1，2 および 3 のみに対して調査した．

まず，全ての実験対象において，少数の共通文字列を読んだ時点でライセンスを特定できるソースファイルの割合と近いカバー率を達成していることがわかる．実験対象 1 では 2 個，実験対象 2 では 4 個，実験対象 3 では 2 個，実験対象 4 では 18 個の共通文字列を利用者が読んだ時点で，ライセンスを特定できるソースファイルのうち，90%以上のソースファイルのライセンスを特定できている．

次に，手法が終了した時点でのカバー率はライセンスを特定できるソースファイルの割合とほぼ同値となっているが，上限には達していない．ライセンスを特定できるソースファイルの割合は実験対象 1 で 86.3%，実験対象 2 で 88.2%，実験対象 3 で 38.3%であるのに対し，手法が終了した時点でのカバー率は，実験対象 1 では 86.3%，実験対象 2 では 88.1%，実験対象 3 では 38.2%であった．

また，図 5 の 10 個目以降のように，各グラフでカバー率の上昇が見られない部分が広範囲にわたって存在する．これらの部分では 2.2 での共通文字列の分類において，2. と 3. に分類される共通文字列が手続きにより表示されていた．

表 2　実験で使用した計算機環境

OS	FreeBSD 6.2RELEASE
CPU	Intel Xeon 5160 3.0GHz × 2
搭載メモリサイズ	3G バイト

3.2.2　評価 2:所要時間の測定

提案手法による対象ファイル数と所要時間の関係は，表 3 の通りである．実験結果より，8000 ファイル程度のソースファイル集合に対して，利用者は 1 時間以内でライセンスを特定することができた．また，ソースファイル数 2000 以上の 3 つの実験対象に対して提案手法を適用した場合，手作業によるライセンスの特定 (図 2, 行 4-14) にかかる所要時間は手法全体の所要時間の 50%以上を占めた．

3.2.3　評価 3:所要共通文字列数の測定

提案手法による対象ファイル数と所要共通文字列の関係は，表 4 の通りである．ソースファイル数が多い場合，手法により提示される共通文字列は多くなっていた．また，ライセンスの特定に使用できなかった共通文字列がその大半を占めていた．例えば，単なる著作権表記や，複数のライセンスに共通する表記が挙げられる．ライセンスの特定に使用できない共通文字列をフィルタリングすることで，利用者の労力を削減することが出来ると考えられる．

4　考察

4.1　本手法の有効性

実験結果において，カバー率はライセンスを特定できるソースファイルの割合とほぼ同値であった．この結果より，提案手法によりライセンスを特定できていないファイルが存在するが，その数はごくわずかであるるため，実用上は問題ないと考えられる．また，全ての実験対象において，4 個程度の共通文字列を読んだ段階で，ライセンスを特定できるソースファイル（実験対象 4 については最終的にライセンスを特定できていた全てのソースファイル）のうち 8 割以上が特定できている．このため，提案手法を用いた場合，ライセンスを特定できるコメントを含まないソースファイルを除く大部分のファイルを少数の共通文字列で特定できている．

提案手法では，利用者は表示された共通文字列全てを読まなくてはならないため，ライセンスを特定できない文字列を読むために多くの労力を割いている．このような共通文字列を機械的に除去することにより，利用者の労力をさらに減らすことができると考えられる．この目的のために，現在，機械学習による文書分類手法 [13] を用いることを考えている．

適用実験では，最大で 8000 ファイル程度のソースファイル集合を実験対象として用いた．この実験対象のソースファイル数は，現在利用可能なソースファイル検索システムが検索対象とするソースファイル数と比べて小さい．具体的には，Koders は約 330000 個，SPARS-J のデモシステム [4] は約 300000 個の Java クラスを検索

表 3　所要時間

	ファイル数	コメントの抽出 (秒)	共通文字列の抽出 (秒)	手作業によるライセンスの特定 (秒)
実験対象 1	554	16.5	25.5	13
実験対象 2	2141	52.3	67.6	660
実験対象 3	2348	42.9	63.5	1856
実験対象 4	7819	193.2	994.8	1428

表 4　所要文字列数

	ファイル数	ライセンスの特定に用いた共通文字列	ライセンスの特定に用いなかった共通文字列
実験対象 1	554	2	2
実験対象 2	2141	9	42
実験対象 3	2348	11	910
実験対象 4	7819	17	659

対象としている [11]．しかし，実験対象は，広く用いられているライセンスを含む様々なライセンスのソフトウェアで構成されており，ライセンスの種類に関しては現実を反映しているといえる．今後，大規模なソースファイル集合に対して手法を適用することにより．より実用的な観点での評価を行うことが必要である．

また，適用実験では提案手法の有効性をライセンス特定の効率，利用者が共通文字列を読む労力，ライセンスの特定にかかる時間の 3 つの観点から評価した．しかし，適用実験は提案手法を利用することが，完全に手作業で行う場合や関連手法に比べ，どのような長所や短所があるか示せていない．そのため，今後の課題として，提案手法と，関連手法や手作業との比較実験を行うべきである．

4.2 本手法の制限

本手法では，1 つのソースファイルに 2 つ以上のライセンスが指定されている場合や，例外規定が付加されたライセンスが指定されている場合には，ライセンスを正しく特定できないことがある．まず，複数のライセンスが指定されている場合，一方のライセンスを指定する記述が含まれていると利用者が判断し，ライセンスを入力した時点で，そのソースファイルはライセンスが特定されたことになるため，他のライセンスを指定する文章が併記されていたとしても，そのソースファイルはライセンス特定の対象となることは無くなってしまう．次に，ライセンスに例外規定が付加されている場合は，付加される前のライセンスの条項の部分だけが共通文字列として検出されてしまい，例外規定の部分が利用者に提示されないままとなってしまう．

複数の同時に指定されているライセンスや例外規定は，もとのライセンスの文字列と連続して出現することが多く，また，共通文字列として抽出されるという特徴がある．そこで，未特定ファイル数が最大となる共通文字列を表示する際，同一ファイル内で前後に存在している共通文字列を同時に提示し，利用者がそれらの共通文字列も読むことにより，上記のような問題に対処することを検討している．

5 関連研究

5.1 ライセンス特定手法

Tunnanen らは正規表現を用いてライセンスを特定する手法を提案している [9]．この手法ではライセンスを指定する記述と適合する正規表現を作成し，各ソースファイルがどの正規表現を適合するかにより，ソースファイルが基づくライセンスを特定する．この手法の利用者はライセンスが特定されていないソースファイルを見て正規表現を作成する．提案手法では手法の利用者が準備を行う必要が無い分，利用者がライセンスの特定を行うのにかかる労力は小さいと考えられる．

5.2 ソースファイル検索システム

提案手法はソースファイル検索システムの運用を支援する手法と位置づけられる．ライセンスに基づく検索が可能なソースファイル検索システムとして，Koders [6] や Google Code Search [5] などがあり，Google Code Search ではコメントまたは COPYING，LICENSE など，特定名前をもつファイルからライセンス情報を取得する．また，Koders や Google Code Search がライセンスを特定する対象となるソースファイル数は多いが，ライセンス特定のカバー率が低い．一方，提案手法はライセンス特定のカバー率が高いが，ライセンスの特定にソースファイルのコメントのみ利用しているため，ライセンスを別のファイルに記述する方法には対応していない．

6 結論と今後の課題

本研究ではソースファイルのコメント中に共通する文字列を用いたライセンス特定支援手法を提案した．オープンソースソフトウェアに含まれるソースファイルの

集合に対する適用実験により，少ない労力でライセンスを特定することが可能であることを示した．今後は，利用者がより小さい労力で正しくライセンスが特定できるよう，手法の改善に取り組む予定である．具体的には，機械学習を用いたライセンスの指定を含まない共通文字列の除去や，１つのソースファイルに複数のライセンスや例外規定が付加されたライセンスを正しく特定できるように，未特定ファイル数が最大となる共通文字列を表示する際，同一ファイル内で前後に存在している共通文字列を同時に提示することに取り組む．また，提案手法をより大規模なソースファイル集合に対して適用可能にし，ソースファイル検索システムに組込むことも重要な課題である．

謝辞 本論文の作成において，多くの助言を頂きました大阪大学大学院情報科学研究科石尾隆氏に深く感謝する．本研究は一部， 独立行政法人日本学術振興会科学研究費補助金萌芽研究 (課題番号：18650006) の助成を受けている．

参考文献

[1] Apache software foundation. http://apache.org.
[2] Asf source header and copyright notice policy. http://www.apache.org/legal/src-headers.html.
[3] Codase. http://www.codase.com/.
[4] demo.spars.info. http://demo.spars.info/.
[5] Google Code Search. http://www.google.com/codesearch.
[6] koders. http://www.koders.com/.
[7] Open source initiative. http://www.opensource.org/.
[8] Sourceforge.net. http://sourceforge.net/.
[9] *Retrieving Open Source Software Licenses*, Proceeding of OSS 2006, 2006.
[10] Christine L. Braun. *Reuse, Encyclopedia of Software Engineering*, Vol. 2. John Wiley & Sons, 1994.
[11] Oliver Hummel and Colin Atkinson, editors. *Using the Web as a Reuse Repository*, Proc. 9th ICSR 2006, 2006.
[12] Toshihiro Kamiya, Shinji Kusumoto, and Katuro Inoue. CCFinder: a multilinguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering*, Vol. 28, No. 7, pp. 654 – 670, 2002.
[13] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, Vol. 34, No. 1, pp. 1–47, 2002.
[14] 横森励士, 梅森文彰, 西秀雄, 山本哲男, 松下誠, 楠本真二, 井上克郎. Java ソフトウェア部品検索システム SPARS-J. 電子情報通信学会論文誌 D-I, Vol. J87-D-I, No. 12, pp. 1060–1068, 2004.